# UNIVERSIDAD CARLOS III DE MADRID

## Escuela Politécnica Superior



Universidad
Carlos III de Madrid

# Desarrollo de un sistema de juego al Tres en Raya para el robot NAO H25


Autor: Nerea Luis Mingueza

Tutor: Daniel Borrajo Millán


25 de junio de 2013

*“ Logic will get you from A to B. Imagination will take you everywhere.”*

Albert Einstein

# *Resumen del Trabajo*

En este trabajo se presenta el desarrollo de un módulo para el robot NAO que le permitirá jugar al “Tres en Raya” con un humano de forma autónoma.

El desarrollo del juego incluye una gran variedad de técnicas de inteligencia artificial como el aprendizaje por demostración, la visión artificial y la búsqueda heurística que le permitirán alcanzar al robot ese grado de autonomía. La finalidad del trabajo es conseguir que el robot NAO aumente el componente social e interactúe cada vez mejor con los humanos mediante juegos sencillos que todos conocemos, pues este trabajo sirve de base para desarrollar otros módulos con otros juegos diferentes.

En la realización del proyecto se ha utilizado el simulador Choregraphe y el API de NaoQi, desarrollados por Aldebaran Robotics para trabajar expresamente con el robot NAO y las librerías de programación OpenCV, para el procesamiento de imágenes, y AIMA, para el algoritmo del juego. El código se ha desarrollado en Python. El proyecto se ha probado sobre un entorno real, con un jugador humano y un robot NAO.

# *Summary*

This thesis presents the design and development of a new module for NAO, the humanoid robot, which allows it to play Tic Tac Toe against a human player in an autonomous way.

The development includes a wide variety of artificial intelligence based techniques such as Learning From Demonstration, Computer Vision and Heuristic Search in order to reach that autonomous behavior on the robot. The aim of this work is to get the robot NAO increase its social component to improve the interaction with humans through simple well-known games. This work serves as the basis to develop other modules with different games.

The code of the Tic Tac Toe module has been developed in Python with the help of Aldebaran's Choregraphe simulator and NaoQi API to work with the robot, OpenCV for image processing and AIMA library for the algorithm of the game. The module has been tested in a real environment where a Tic Tac Toe game was played by a human and a robot NAO.

# *Agradecimientos*

En primer lugar me gustaría agradecer a mi familia todo el esfuerzo, el apoyo y la confianza que han depositado en mi desde el primer día que entré en la universidad. Sin ellos estar donde estoy hubiera sido imposible. Gracias por todo.

Agradecer también a Daniel Borrajo, el tutor de este trabajo y la persona que ha sabido sacar lo mejor de mí en cada momento. Gracias por todo el tiempo que me has dedicado, desde que inicié las prácticas en tercer curso hasta el día de hoy.

A los integrantes del *Planning and Learning Group* de la Universidad, porque todos y cada uno de vosotros me habéis acogido como a una más, habéis hecho que aprenda una cantidad increíble de cosas y habéis despertado aún más mi interés por la investigación. Dentro de poco espero poder enseñaros yo también alguna :) . Por supuesto, un agradecimiento especial a todos mis compañeros del laboratorio: a Moisés y Ezequiel por enseñarme todo lo relacionado con la robótica y los NAO, que sin ellos hubiera sido imposible comenzar a trabajar los primeros meses, a Isa, Jesús, Álvaro, Vidal y Sergio por todas esas horas que hemos compartido de risas, nervios, *deadlines*, etc. Sois increíbles.

A Daniel Pérez, porque fue mi profesor de programación pero también con el que compartí gran parte de mi tiempo durante el primer curso. Con tu trabajo conocí la inteligencia artificial, leí *papers*, aprendí sobre las emociones en los agentes artificiales (¡y todo esto en primer curso!) y gracias a tí conocí y entré en el PLG. Gracias por tu confianza.

También quiero agradecer a los compañeros con los que he cursado esta carrera, por todos los ratos buenos que me han hecho pasar, porque desde que esto dio comienzo con Álvaro he compartido mi tiempo con muchas otras: Sara, Ana, Javi, Miguel, Jenifer, Jessica, ... ¡LegaTech! y muchos otros más. Porque los días duros de "9 a 9" en la universidad, el estudio previo a exámenes finales y todo hubiera sido mucho más difícil de superar si no hubiérais estado ahí.

Por supuesto, a la gente de Delegación de Estudiantes, porque día a día intentamos que la Universidad mejore otro poquito más. Vosotros me habéis enseñado a dialogar y sobre todo a escuchar, a saber interpretar distintos puntos de vista y a trabajar en equipo. Espero que sigáis siendo tan buenos en años sucesivos.

A Belén, la subdirectora del grado, porque has sido capaz de animarme, calmarme y centrarme cuando lo necesitaba. Porque sin tí el T3chFest, o lo que es lo mismo, una de las mejores experiencias de mi vida, no hubiera sido posible. Gracias por tu entusiasmo y tus ganas de escuchar a todos y cada uno de nosotros.

Baldo, Josito, muchas gracias por todos los consejos y por todos los buenos ratos que me habéis hecho pasar dentro y fuera de la Universidad. Habéis sido indispensables. Para mí habéis sido el modelo a seguir durante todos estos años y de verdad que os doy las gracias por ello.

Por último quiero agradecer a Rodry todo su apoyo, su confianza ciega en mí, sus ánimos que me hacen siempre seguir adelante y su paciencia, porque sé que a veces aguantarme es complicado. Ahora toca comenzar una nueva etapa y espero que tú sigas ahí conmigo. Gracias de verdad por todo este tiempo.

# Índice general

# Índice de figuras

# Índice de tablas

*A Chiqui.*

# Capítulo 1

# Introducción

Este primer capítulo está dividido en tres secciones. En la sección 1.1 se presentan los motivos por los que se decidió llevar a cabo el trabajo. En la sección 1.2 se enuncian los objetivos de éste. Por último, en la sección 1.3 se explica la estructura del presente documento.

## 1.1. Motivación

En una sociedad en la que la tecnología avanza con una velocidad vertiginosa, la continua aparición de todo tipo de dispositivos (*smartphones*, computadoras, videoconsolas, televisiones 3D etc.) hace que la innovación sea uno de los pilares clave para que las empresas sigan recibiendo grandes cantidades de beneficios. Cada vez queremos que nuestra vida sea más cómoda y placentera y esto va ligado al avance de la tecnología. Ya nos hemos acostumbrado a estar rodeados de dispositivos electrónicos y de estar conectados a Internet constantemente. De esta forma, es habitual encontrar en las casas lavadoras y robots de cocina inteligentes, robots que limpian el polvo, recomendaciones de productos personalizadas a través de Internet, juguetes que se comportan de una forma u otra en función del usuario etc. Pero no sólo estamos rodeados de dispositivos electrónicos en casa, también están presentes en prácticamente cualquier sitio al que acudamos: hospitales, centros comerciales, intercambiadores etc.

La mayoría de estos dispositivos, objetos o servicios web que se han mencionado llevan por detrás una gran desarrollo de técnicas pertenecientes a la inteligencia artificial. Es la única forma de que se modifique el comportamiento de un programa evitando que el robot que limpia el polvo se choque con las esquinas, de que se puedan personalizar correos electrónicos extrayendo previamente perfiles de personas o de que un agente virtual pueda interactuar con una persona por voz o mensajes a través de una interfaz.

Desafortunadamente, en el área de la robótica, el coste y complejidad de desarrollo de un robot de estas características es bastante alto. El *hardware* evoluciona mucho más rápido que el *software* y esto provoca que haya robots muy robustos pero torpes o poco inteligentes. Habitualmente, los robots se diseñan para resolver un número finito de tareas combinables de forma individual o colectiva pero, por ejemplo, algo tan sencillo como que un robot bípedo sea capaz de levantarse y sentarse de forma autónoma puede ser un gran obstáculo para el desarrollo de un programa más complejo. Las líneas de investigación de la robótica están centradas principalmente en disminuir esa brecha tecnológica para poder conseguir un comportamiento inteligente o similar al de un humano, pero no en todos los casos es posible. A día de hoy los robots que más se acercan a esto son los robots sociales, por sus características comunicativas con el entorno y su forma de actuar.

Esta idea de reducción de brecha tecnológica entre *hardware* y *software* es la que se pretende estudiar en este trabajo para proponer alguna solución. Un robot puede no estar destinado a realizar ciertas tareas pero ¿es imposible que las realice? Si se dispone de los medios y las técnicas adecuadas como las que ofrece la inteligencia artificial seguramente será capaz de aprender. De esta forma, aunque sus acciones no sea tan perfectas como las de otro robot especializado en el campo, se dispondrá de un robot mucho más versátil. Esto vuelve a cobrar importancia a la hora de trabajar con robots sociales, pues es muy interesante que sepan desarrollar y resolver gran cantidad de tareas y situaciones diferentes para mostrar un mayor acercamiento con los seres humanos.

## 1.2. Objetivos generales

Este trabajo recoge el desarrollo de un sistema realizado para el robot humanoide NAO H25. El objetivo de dicho sistema es que el robot sea capaz de jugar, de forma autónoma, al *Tres En Raya* sobre un tablero representado en una pizarra.

En primer lugar, el robot aprenderá a dibujar dos símbolos diferentes sobre un tablero. Después, será capaz de identificar en qué situación del juego se encuentra y la casilla que ha escogido su rival. Finalmente será capaz de tomar la decisión sobre qué casilla pintar en cada turno.

Hay que destacar que este robot no está especializado en tareas de precisión como la de dibujar, de ahí la dificultad añadida al desarrollo del módulo.

## 1.3. Estructura del documento

El presente documento se ha estructurado en capítulos. A fin de ofrecer una visión global del documento, a continuación se enuncia el contenido de cada uno de estos capítulos.

En el capítulo 1 se relata cuáles han sido las motivaciones por las que se decidió realizar este trabajo, los objetivos del trabajo y la estructura que sigue del presente documento.

En el capítulo 2 se realiza un estudio de la base teórica relacionada con el trabajo desarrollado.

En el capítulo 3 se describe el alcance del trabajo y se explica detalladamente la fase de análisis y diseño del trabajo. También se incluye una explicación exhaustiva del sistema, analizando en profundidad cada componente.

En el capítulo de 4 se muestra el resultado de las pruebas a las que se ha sometido el sistema, las cuales verifican que su funcionamiento es el esperado y el correcto.

En el capítulo de 5 se presenta la metodología utilizada durante el desarrollo del trabajo, la información relativa a la planificación de las tareas y la información sobre presupuesto total y desglosado del trabajo.

En el capítulo de 6 se enuncian las conclusiones extraídas de la realización del trabajo y se proponen varias funcionalidades y nuevas líneas de investigación que mejorarían y complementarían el módulo implementado.

Finalmente, el capítulo 7 contiene un resumen del trabajo realizado en inglés.

# Capítulo 2

# Estado del arte

En este capítulo se recogen las diferentes investigaciones, técnicas y herramientas que han sido de utilidad para el desarrollo del proyecto. En primera instancia se habla de la historia de la robótica, de los distintos sistemas de control de robots y las diferentes formas de programarlo. La teleoperación, el aprendizaje por demostración o la lógica difusa son algunos de los temas que se recogen en estos apartados.

Después, se explica el funcionamiento del algoritmo de búsqueda heurística MiniMax y la poda Alfa-Beta que ayudarán a implementar el algoritmo de juego del Tres en Raya.

Por último se habla de algunas técnicas englobadas en la visión artificial que ayudan a reconocer y procesar la información del entorno.

## 2.1. Historia de la robótica

A lo largo de la historia, el ser humano ha demostrado un interés constante por diseñar y construir artefactos o sistemas que sean capaces de realizar parte del trabajo diario. Aunque hasta el año 1921 no se utilizó el término "robot" para referirse a este tipo de sistemas, sí que se encuentran referencias de éstos con las palabras "máquina" y "autómata", pues se diseñaban y construían con los mismos fines.

En la antigüedad, la civilizaciones egipcia y griega sorprendieron con los primeros sistemas automatizados. Los sacerdotes egipcios construyeron brazos mecánicos y las unieron a las estatuas de sus dioses. Por otra parte, los griegos construyeron sistemas hidráulicos para dotar de movimiento a las estatuas.

La palabra robot aparece por primera vez en la obra teatral de ciencia ficción *R.U.R. (Rossum's Universal Robots )*. Su autor es Karel Čapek (1890-1938), de nacionalidad checa. El hermano de

Karel, llamado Josef, sugirió al primero utilizar la palabra checa “robota”, cuyo significado es *“Trabajo forzado, servidumbre”*, para referirse a los androides de los que hablaba en el libro. Por otro lado, Josef ya utilizó en 1917 la palabra checa “automat” en su obra *Opilec* para referirse a estos androides. Los hermanos fueron los que decidieron reemplazar “automat” por “robot” en la obra *R.U.R.* Debido al éxito del libro de Karel, los traductores adaptaron el término “robota” a “robot” en la edición inglesa, así es como dicha palabra comenzó a ganar popularidad.

Isaac Asimov (1920 - 1992), escritor de renombre en el campo de la ciencia ficción y la divulgación científica, publicó en 1942 el libro *Runaround* en el que se enunciaban ”Las tres leyes de la robótica”:

- Un robot no puede hacer daño a un ser humano o, por inacción, permitir que un ser humano sufra daño.
- Un robot debe obedecer las órdenes dadas por los seres humanos, excepto si estas órdenes entrasen en conflicto con la Primera Ley.
- Un robot debe proteger su propia existencia en la medida en que esta protección no entre en conflicto con la Primera o la Segunda Ley.

De esta forma, mediante sus libros, Asimov difundió al resto del mundo la idea de que cualquier robot se ha diseñado para cumplir con una serie de tareas y ser fiel al ser humano. Estas tres leyes fueron tan bien recibidas que la pura ficción se convirtió en realidad y Asimov ha sido referenciado en numerosos artículos científicos, películas y libros.

Teniendo en cuenta la evolución de la robótica desde los inicios hasta la actualidad se distinguen cuatro generaciones de robots [1]:

- Primera generación: Robots manipuladores. Están diseñados para realizar una serie de tareas, previamente programadas por lo que no les afecta el entorno en el que se encuentren; no son sensibles a las variaciones. Los sistemas de control que llevan implantados son muy simples y se conocen como sistemas de lazo abierto [2], pues en función de una misma entrada se produce una misma salida que no afectará en el futuro al resto de entradas; no existe ningún tipo de retroalimentación en el sistema. Algunos ejemplos de robots de esta generación son los brazos robóticos de uso industrial. La figura 2.1 muestra un robot que pertenece a los de primera generación. Su cometido es desplazarse siguiendo la línea.

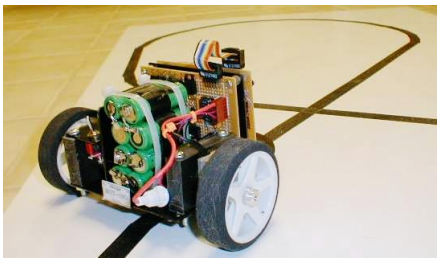


FIGURA 2.1: Robot de 1ª Generación. (Fuente)

- Segunda generación: Robots enfocados a la teleoperación. En esta generación los robots eran capaces de memorizar y repetir una serie de pasos realizados por un operador humano. De esta forma el robot era capaz de realizar las mismas acciones que un humano sin que éste corriese ningún tipo de riesgo. El sistema de control sigue siendo de lazo abierto. La técnica en la que se trabaja con estos robots es conocida como teleoperación. La figura 2.2 muestra un robot que pertenece a los de segunda generación, su cometido es realizar los mismos movimientos que el operador.

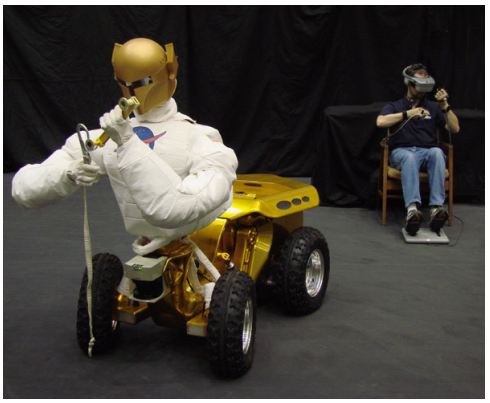


FIGURA 2.2: Robot de 2ª Generación. (Fuente)

- Tercera generación: Robots sensibles al entorno a través de sensores. Estos robots estaban dotados de una gran cantidad de sensores que les permitieron obtener información del entorno para así poder realizar de una forma u otra cada una de las ejecuciones a través de unos componentes llamados actuadores. Ahora los robots sí que contaban con un sistema de retroalimentación mediante el cual podían saber si su ejecución había sido exitosa o no. Estos sistemas se llaman de lazo cerrado [2]. Las nuevas ejecuciones dependerán tanto de resultados de ejecuciones anteriores como del entorno en el que se encuentre el robot y la entrada que reciba el sistema. Se empieza a trabajar en técnicas de planificación y replanificación de tareas. La figura 2.3 muestra un robot que pertenece a los de tercera generación. Su cometido es desplazarse esquivando los obstáculos.

FIGURA 2.3: Robot de 3ª Generación. (Fuente)

- Cuarta generación: Robots inteligentes. Esta generación corresponde a la de los robots más avanzados en cuanto a tecnología. Los componentes de los robots se han mejorado, ahora tienen cámaras y se dispone de sensores mucho más complejos que permiten dotar de gran cantidad de información al robot consiguiendo que éste ejecute acciones complicadas con éxito en tiempo real. Se utilizan técnicas de visión artificial, de planificación, de aprendizaje, de toma de decisiones etc. todas estrechamente relacionadas con la inteligencia artificial para conseguir que el robot sea autónomo. La figura 2.4 muestra un robot que pertenece a los de cuarta generación.

FIGURA 2.4: Robot de 4ª Generación. (Fuente)

A pesar de que aquí se haya mostrado esta clasificación existen innumerables tipos de clasificación de robots [3].

## 2.2. Diferentes aproximaciones de control de un robot

En esta sección se recogen los tres tipos de sistemas de control que pueden encontrarse en un robot en la actualidad: programados, teleoperados y autónomos [4]. Como se puede observar, los sistemas de control se han ordenado en función del grado de autonomía que le aportan al robot.

### 2.2.1. Programados

Los robots programados son aquellos que están compuestos por un circuito lógico o un circuito integrado en una placa base. Son los robots más básicos y sencillos que existen. El sistema siempre va a comportarse de la misma manera si recibe una misma entrada, independientemente de cuál sea el entorno que le rodea. Los requisitos de cómo y cuándo poner en marcha un robot programado los establece y controla el usuario.

Algunos ejemplos comunes de este tipo de robots son los que siguen una línea (figura 2.5) o un foco de luz. Son robots expertos en su tarea pero muy sensibles a los cambios en el entorno.

FIGURA 2.5: Robot programado. (Fuente)

### 2.2.2. Teleoperados

La técnica de la teleoperación (trabajo a distancia) [5] se comenzó a utilizar en la segunda generación de robots, cuando los humanos se dieron cuenta de que en algunas profesiones se realizaban acciones arriesgadas y era complicado acceder a ciertos lugares. La finalidad de esta técnica es reducir el riesgo de peligrosidad que existe a la hora de que un humano realice una determinada acción y, en su lugar, controlar a un robot a distancia para que lo haga. En la figura 2.6 se muestra un ejemplo real donde se aplica la teleoperación.

Para llevar a cabo dicha técnica hay que identificar primero al maestro, el humano, y al esclavo, el robot, y establecer un medio de comunicación entre ambos. Algunas dificultades que surgen a la hora de desarrollar este proceso es la coordinación humano-robot [6], el leve retraso que sufre el sistema cuando el humano transmite cada acción hasta que el robot la ejecuta, la estabilidad del robot, lo real que resulte el escenario ficticio para el humano y la precisión del robot a la hora de ejecutar los movimientos.


FIGURA 2.6: Ejemplo de teleoperación. (Fuente)

Durante la teleoperación el robot no es consciente de lo que está haciendo y de si lo está haciendo bien o mal, simplemente cumple el papel de marioneta. El humano es, entonces, el que debe conseguir ese grado de sensibilidad frente a lo que está viendo el robot e idear la forma de realizar la tarea. Sí que podrá observar los sensores del robot y las cámaras que le proporcionarán toda la información. Igualmente, se pueden utilizar, siempre que se incluyan en el sistema de teleoperación, todas las ayudas que la arquitectura del robot proporcione: sistemas de equilibrio, de seguridad, comportamientos predefinidos para levantarse del suelo etc.

A parte del maestro y el esclavo, los elementos imprescindibles que forman parte de la teleoperación son los siguientes:

- Interfaz: Es la herramienta que utilizará el maestro para mover al robot a distancia. Algunos ejemplos son un smartphone, un joystick o una interfaz hombre-máquina a través de una cámara estereoscópica que sea capaz de calcular la profundidad de la escena para poder medir distancias.
- Canal de comunicación: Para que todos los movimientos realizados a través de la interfaz lleguen al robot, es necesario establecer un canal de comunicaciones inalámbrico o cableado desde el computador hasta el robot.

El esquema completo de teleoperación queda reflejado en la figura 2.7.

### 2.2.3. Autónomos

Para que un robot tenga control propio y pueda ser autónomo hay unos requisitos mínimos que se deben cumplir: obtener y procesar información del entorno, realizar movimientos seguros con

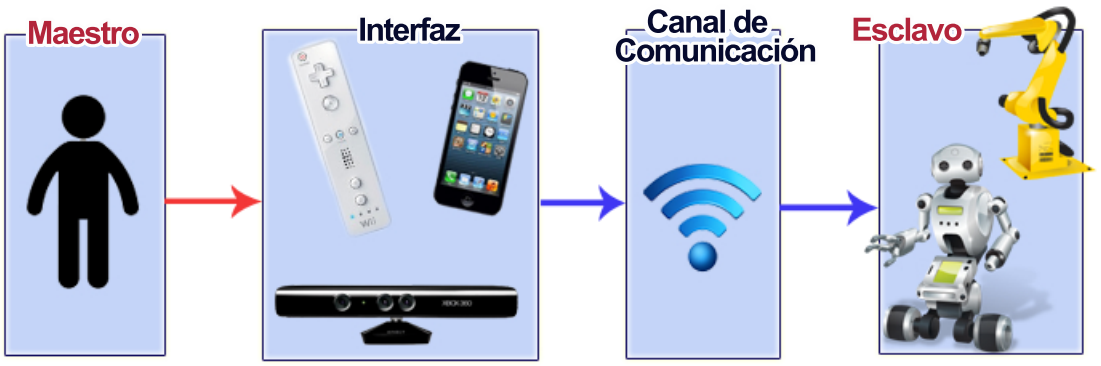



FIGURA 2.7: Esquema de teleoperación.

cada articulación, recuperarse frente a un imprevisto y evitar dañar a una persona o a sí mismo. Adicionalmente, un robot autónomo también puede aprender de cada una de sus ejecuciones anteriores para poder mejorar e innovar sus comportamientos.

Para llevar a cabo este tipo de tareas, existen dos aproximaciones de control: reactiva y deliberativa [7]. También se pueden combinar construyendo un sistema híbrido.

La principal diferencia entre ambas aproximaciones reside en el proceso interno que siguen a la hora de dar una respuesta ante los estímulos del entorno. En los siguientes apartados se hablará con más detalle de cada una de estas aproximaciones.

#### 2.2.3.1. Control reactivo

Los sistemas basados en un control reactivo destacan por la gran rapidez a la hora de dar una respuesta ante la información recibida del entorno a través de los sensores del robot, de ahora en adelante estímulos. Para saber qué respuesta dar, tienen en cuenta los cambios del entorno, pero éstos deben estar previamente definidos en el sistema. A este tipo de sistemas se les conoce como estímulo/respuesta porque el propio robot nunca cambiará la respuesta ante un mismo estímulo, de ahí la simpleza y la rapidez a la hora de tomar decisiones.

La principal desventaja de este tipo de control es que no es capaz de reaccionar correctamente ante comportamientos inesperados; son ciegos con respecto al futuro. Por otro lado, la principal ventaja es que los robots que implementan este tipo de sistemas son grandes especialistas en sus tareas, pues aunque el entorno cambie, siempre será muy similar y los resultados serán rápidos.

#### 2.2.3.2. Control deliberativo

Los sistemas basados en un control deliberativo invierten gran parte del tiempo de procesamiento en la toma de decisiones. La arquitectura es mucho más compleja, pues habitualmente sigue una estructura jerárquica o está dividida en varias capas que se comunican entre sí.

En este tipo de sistemas de control existirá un módulo que se encargará de analizar y procesar toda la información recibida por los sensores para transmitirla al al módulo de toma de decisiones. Éste tendrá en cuenta los objetivos que debe conseguir evaluando hasta cuáles puede alcanzar el robot dado el estado en el que se encuentra. Finalmente la decisión tomada será traducida a los actuadores que ejecutarán la acción sobre el entorno.

Un sistema de control deliberativo es capaz de aprender del entorno y de variar su comportamiento. Pero, si el entorno es demasiado cambiante o si el propio sistema tarda un tiempo considerable en tomar una decisión, la acción que ejecutarán los actuadores puede no ser la correcta porque los cálculos estarán obsoletos. Esta es una de las principales desventajas.

## 2.3. Formas de programar un robot

En esta sección se recogen varias técnicas útiles para programar un robot. Cada una de ellas tiene unas características diferentes que se adecuarán en mayor o menor grado al objetivo final que se persiga con el robot.

### 2.3.1. Ecuaciones diferenciales

En la dinámica de un robot móvil están relacionadas las coordenadas en las que se encuentra, la velocidad, la aceleración, las diferentes inercias, las fuerzas que ejercen sobre el robot, etc. Por ello, una de las opciones que un desarrollador se puede plantear es desarrollar un sistema de ecuaciones diferenciales basado en un modelo cinemático que permita al robot calcular en tiempo real los valores de velocidad, aceleración y desplazamiento que puede alcanzar ejecutando una determinada acción o conjunto de acciones. Un factor muy importante a tener en cuenta es el tiempo, pues se deberán generar pares de valores en los que uno de ellos sea el tiempo para conocer la duración de cada pequeño movimiento con el fin de poder alcanzar la meta propuesta. Precisamente por el factor tiempo y por la necesidad de obtener los cálculos en tiempo real, el modelo matemático a desarrollar no es trivial porque las ecuaciones diferenciales son de segundo orden y el problema pertenece al área conocida como cinemática inversa. Hay cuatro teoremas imprescindibles que permiten resolver gran parte de dicho problema y son las siguientes:

- La formulación del modelo dinámico de Euler-Lagrange
- La formulación del modelo dinámico de Newton-Euler
- La teoría de Lyapunov [8] para conseguir el punto de equilibro de un controlador
- El modelo dinámico para un robot de n grados de libertad desarrollado por Spong [9]

Por otro lado, se enuncian brevemente a continuación algunos de los sistemas de control basados en ecuaciones diferenciales más utilizados:

- Control Proporcional-Derivativo: Es el más sencillo y garantiza que el error del modelo dinámico en el robot es mínimo cuando se desplaza y cero cuando permanece quieto. Es el control clásico de los manipuladores robóticos.
- Control Adaptativo [8]: Se encarga de ajustar parámetros y señales de control en función de las circunstacias que percibe del entorno. Es muy útil cuando no se conoce completamente el modelo dinámico del robot.
- Control por Linealización Exacta [10]: El objetivo es el mismo que los otros dos tipos de control sólo que en este se aplican transformaciones espaciales al robot.

### 2.3.2. Lógica difusa

La lógica difusa o *Fuzzy Logic* [11] fue ideada para recoger y trasmitir el conocimiento respecto a un entorno en el que influye en gran medida la forma de verlo y lo relativo de la situación; es decir, es capaz de reflejar la incertidumbre e imprecisión de un concepto.

Un sistema de control basado en lógica difusa lleva implementado un conjunto de reglas del estilo “Si *precondición* entonces *consecuencia*”, donde la precondición y la consecuencia, habitualmente, van acompañadas de un adverbio que hace de cuantificador aportando esa visión relativa de la situación. Una sentencia difusa sería “Si agua algo caliente entonces cambiar poco la temperatura”. En la figura 2.8 se puede ver gráficamente esa sentencia representada en conjuntos borrosos.

Dado que las sentencias son puramente lógicas, es habitual en ellas encontrar negaciones, uniones, intersecciones y demás componentes lógicos sobre todo en la precondición de la sentencia.

El funcionamiento de un sistema de control difuso comienza cuando se reciben una o varias entradas. Éstas se corresponden con la precondición de alguna de las sentencias lógicas implementadas desencadenando así la consecuencia de la sentencia. La activación de reglas es llevada a cabo por el motor de inferencia del sistema.

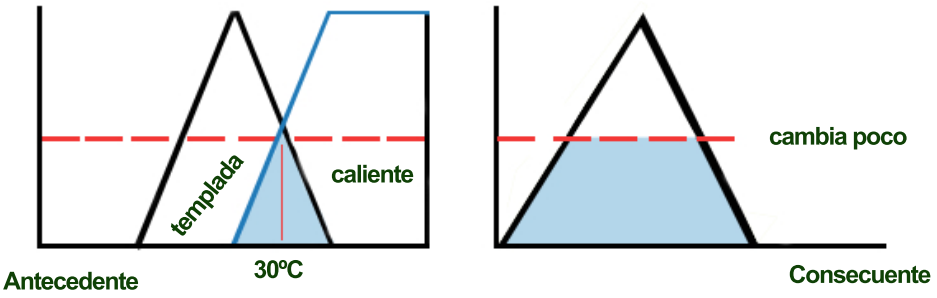



FIGURA 2.8: Ejemplo de conjunto difuso.

Si las entradas son numéricas habrá que traducirlas a entradas difusas mediante el sistema conocido como *Fuzzyfication*. Lo mismo ocurre si se desea que la salida sea numérica, sólo que se aplicará el proceso contrario: *Defuzzyfication*. Ambos procesos consisten en establecer unos intervalos numéricos que se correspondan con los cuantificadores habilitados para cada sentencia. A modo de resumen, la figura 2.9 incluye un esquema del bucle que implementa cualquier sistema basado en lógica difusa.

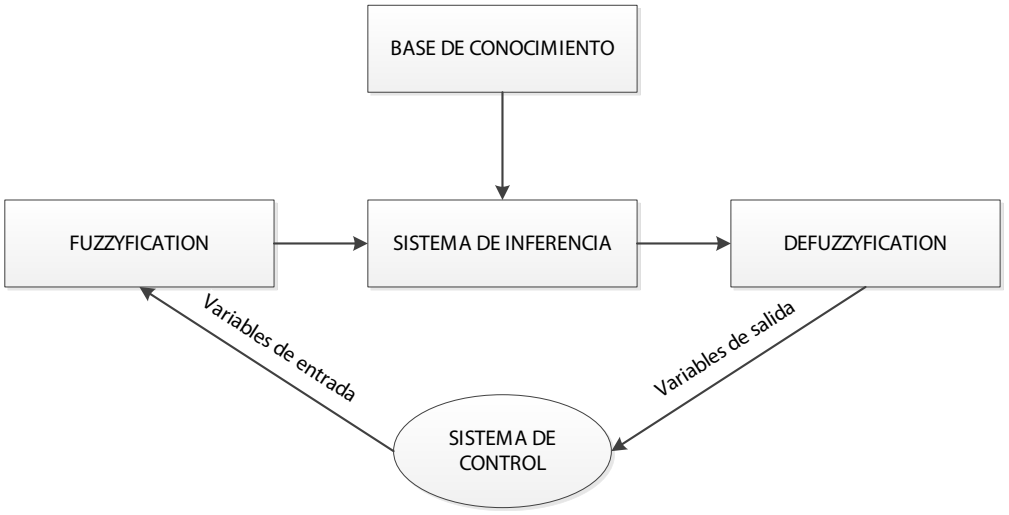



FIGURA 2.9: Bucle de control en lógica difusa.

Aplicado a la robótica, la lógica difusa es muy útil para ayudar a la toma de decisiones en el módulo de control del robot. Gracias a la simplicidad de las operaciones es una técnica muy rápida y funciona muy bien en entornos en los que la precisión no es demasiado relevante. Algunas implementaciones de lógica difusa en robots son el control de navegación [12] [13] y el rol de futbolista [14].

### 2.3.3. Aprendizaje por demostración

El objetivo del aprendizaje por demostración (LfD, Learning from Demonstration) es conseguir que un robot ejecute comportamientos nuevos sin necesidad de programarlos. La función del humano es guiar al robot para que reproduzca paso a paso la tarea que se desea que ejecute después de forma autónoma. Por este motivo, no es necesario que el humano tenga conocimientos de programación. El rol que asume el humano es el de profesor. En la figura 2.10 se puede ver gráficamente en qué consiste esta técnica.

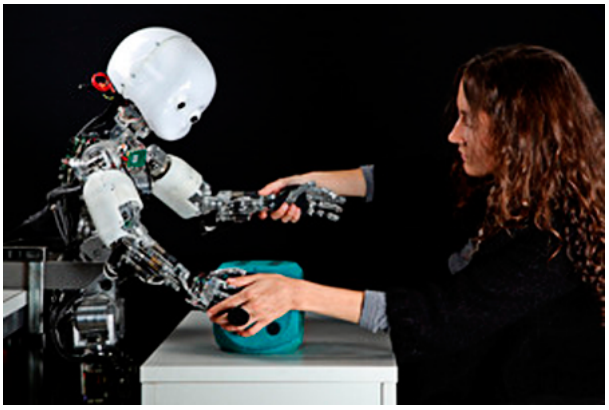


FIGURA 2.10: Aprendizaje por demostración. (Fuente)

Para que esta técnica se desarrolle con éxito, el operador deberá recoger un número significativo de instancias que le permitan al robot generalizar la tarea en un futuro; es decir, ejecutar la tarea en diferentes condiciones pero siempre con éxito. Las dificultades que se encuentran en esta técnica residen principalmente en la traducción de movimientos del operador al robot, pues la reproducción de éstos nunca será exacta. Si el error en estas traducciones es muy alto será mejor utilizar otra técnica que proporcione mejores resultados. Es imprescindible que el robot almacene la mayor cantidad posible de información que perciba del entorno y de él mismo.

Es habitual que el aprendizaje por demostración se encuentre acompañado de otras técnicas como el aprendizaje por refuerzo. De esta forma los movimientos aprendidos mediante la primera técnica, se mejorarán y optimizarán con la segunda indicando al robot en qué ocasiones ha realizado mejor la tarea.

## 2.4. Algoritmo MiniMax

El algoritmo MiniMax [15] [16] es un algoritmo de búsqueda heurística en profundidad que se aplica a juegos de dos jugadores en los que cada uno conoce las posibles estrategias del adversario, ya que la información sobre el estado del juego es completa. El fin que se persigue es minimizar la pérdida máxima de cada jugada teniendo en cuenta que el adversario escogerá la

jugada que peor te beneficie. Algunos juegos muy conocidos con los que se puede utilizar el algoritmo MiniMax son el Ajedrez, el Go, las Damas, el Tres en Raya etc.

Para desarrollar el árbol de búsqueda primero hay que identificar los siguientes elementos:

- Jugador MAX: será la persona que quiere utilizar el algoritmo.
- Jugador MIN: será el contrincante de MAX.
- Función de utilidad: para que el algoritmo pueda tomar una decisión sobre qué jugada escoger hay que evaluar la validez de cada estado posible del juego mediante una función de evaluación.

A la hora de realizar la expansión del árbol de búsqueda hay que tener en cuenta que cada nivel del árbol corresponderá al jugador MAX o a su adversario MIN, por lo que en los nodos MAX, el cometido será elegir el estado que maximice la función de utilidad y en los nodos MIN, el estado que la minimice.

Cuando el algoritmo alcance un estado ganador para MAX, la función de utilidad devolverá el valor $+\infty$. Si por el contrario obtiene un estado ganador para MIN, devolverá $-\infty$. El empate se representa con un cero. Adicionalmente habrá que establecer un límite de niveles a explorar, pues siendo Minimax un algoritmo de búsqueda en profundidad hay que evitar perderse en el árbol poniendo un límite. El Algoritmo 1 describe el pseudocódigo del algoritmo Minimax.

La figura 2.11 muestra el árbol generado por el algoritmo y la elección final del camino a seguir para un ejemplo de árbol. Siendo los nodos azules los nodos MAX y los rojos, MIN, se establece la profundidad máxima ($p$) en tres niveles y se expande el árbol hasta alcanzar dicha profundidad. Como los nodos terminales son nodos MIN y sus respectivos nodos padre son MAX, habrá que escoger el que maximice el valor comenzando por el nodo terminal situado más a la izquierda. Los nodos terminales son los únicos que poseen valor numérico inicialmente. Después, estos valores se van propagando hacia el nodo raíz. El valor final que devolverá el algoritmo será el del nodo raíz, que en el caso de la figura 2.11 se ha escogido por ser el mayor de los nodos hijos MIN. Estas cantidades, a su vez, se obtuvieron de los nodos hijo del siguiente nivel y así sucesivamente hasta alcanzar los nodos hoja.

## 2.5. Algoritmo de poda Alfa-Beta

El algoritmo de poda Alfa-Beta [16] ayuda a reducir el número de nodos a expandir y evaluar del árbol que genera el algoritmo MiniMax sin alterar la solución final. Se utilizan dos parámetros, $\alpha$ y $\beta$, que irán actualizándose en función de las expansiones de nodos del árbol de búsqueda.

**Algorithm 1** Algoritmo MiniMax

**Require:** $nodo, p \in \mathbb{N}$.
**Ensure:** Valor de la función de utilidad de $nodo$.

```
if esHoja(nodo) or p = 0 then
    return funcionEvaluacion(nodo)
end if
if nodo es nodo MAX then
    x ← −∞
    for nodo_hijo en nodo do
        x_aux ← minimax(nodo_hijo, p − 1))
        if x_aux > x then
            x ← x_aux
        end if
    end for
    return x
end if
if nodo es nodo MIN then
    x ← ∞
    for nodo_hijo en nodo do
        x_aux ← minimax(nodo_hijo, p − 1))
        if x_aux < x then
            x ← x_aux
        end if
    end for
    return x
end if
```

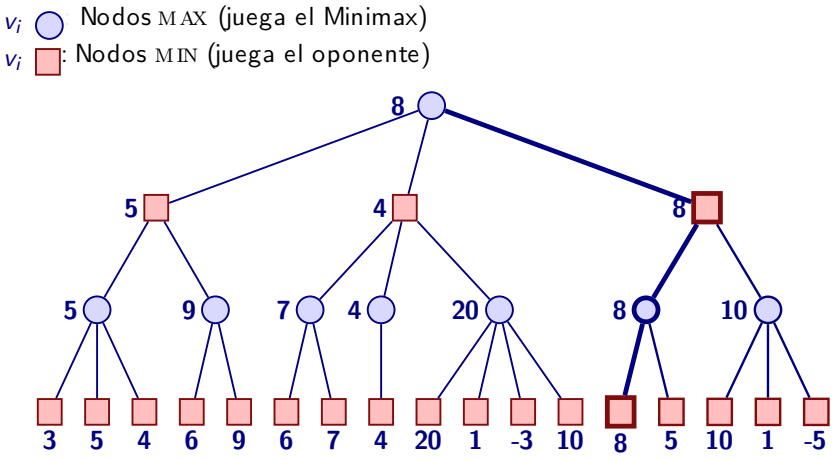



FIGURA 2.11: Resultado de algoritmo Minimax. (Fuente)

- $\alpha$ equivale al valor que posee la mejor opción que se ha encontrado en el camino para el jugador MAX. Este parámetro sólo se actualiza en los nodos MAX.
- $\beta$ equivale al valor que posee la mejor opción que se ha encontrado en el camino para el contrincante MIN. Este parámetro sólo se actualiza en los nodos MIN.

Los valores iniciales de Alfa y Beta son $-\infty$ y $\infty$ respectivamente. Estos valores se actualizan en los distintos nodos del árbol en sentido ascendente; es decir, desde los nodos terminales hasta el nodo raíz.

Siendo $p$ el nivel de profundidad en el que nos encontremos, para podar una rama que comienza en un nodo MAX se debe cumplir que $\alpha_p \geq \beta_{p-1}$. Por otro lado, para podar una rama que comienza en un nodo MIN se debe cumplir que $\beta_p \leq \alpha_{p-1}$. El algoritmo 2 muestra el pseudocódigo de la poda Alfa-Beta.

**Algorithm 2** Algoritmo Alfa-Beta

**Require:** $nodo, p \in \mathbb{N}$.
**Ensure:** Valor de la función de utilidad de $nodo$.
  **if** $nodo_{terminal}$ or $p = 0$ **then**
    **return** valor de la función de utilidad del $nodo_{terminal}$
  **end if**
  **for** $nodo_{hijo}$ en nodo **do**
    $\alpha \leftarrow max(\alpha, -alfabeta(nodo_{hijo}, p-1, \beta, \alpha))$
    **if** $\beta \leq \alpha$ **then**
      break
    **end if**
    **return** $x$
  **end for**

Finalmente, tomando como referencia el árbol generado en la figura 2.11, se aplica ahora al mismo la poda Alfa-Beta quedando el árbol como se refleja en la figura 2.12.

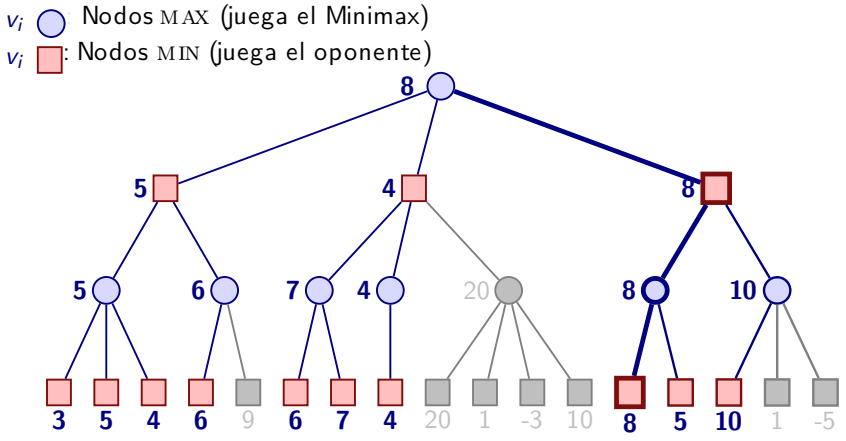



FIGURA 2.12: Resultado de algoritmo Alfa-Beta. (Fuente)

## 2.6. Visión artificial

La visión artificial es un área de la inteligencia artificial dirigida al reconocimiento y procesamiento de imágenes que permiten a un robot o agente analizar el entorno y comprender qué está pasando a través de una imagen o vídeo que ha sido capturado con su propia cámara o que ha sido recibido del exterior.

Mediante la visión artificial se pueden identificar objetos, caras humanas, trazar caminos para alcanzar un objetivo, comparar los cambios que ha habido en el entorno entre dos instantes de tiempo, etc.

De todo lo mencionado anteriormente, en este trabajo se va a profundizar en las técnicas que permiten comparar dos imágenes para detectar los cambios del entorno en dos instantes de tiempo diferentes. Primero se hablará del binarizado y después de los histogramas.

### 2.6.1. Binarizado

La técnica del binarizado de imagen es muy útil para distinguir un objeto del resto del entorno en función de un umbral de color. En primer lugar, hay que transformar la imagen en color a una de escala de grises. Después hay que escoger el umbral de la imagen, que decidirá qué *píxels* obtienen el color blanco (1) y cuáles el negro (0).

En la figura 2.13 se muestra un ejemplo del resultado de aplicar el binarizado sobre una imagen de tres botellas de agua. En este caso, se habría escogido como umbral el color azul, quedando todos los colores similares convertidos al color negro. Como aplicación en la vida real, esto se utiliza para saber rápidamente si una botella se ha rellenado correctamente y hasta la altura indicada.

### 2.6.2. Histogramas

Los histogramas son diagramas de barras que ofrecen la información relativa al número de *píxels* de luminosidad dada una imagen.

Si dicha imagen se encuentra en escala de grises, se obtendrá un único histograma como salida que albergará el rango de colores negro, gris y blanco de forma ascendente en el eje de abcisas. Si por el contrario la imagen es en color (RGB) se obtendrán tres histogramas: uno para el canal rojo, otro para el verde y el último para el azul.

Para que el histograma cobre sentido, el sumatorio de los valores de las barras deberá ser equivalente al número de pixels total de la imagen.

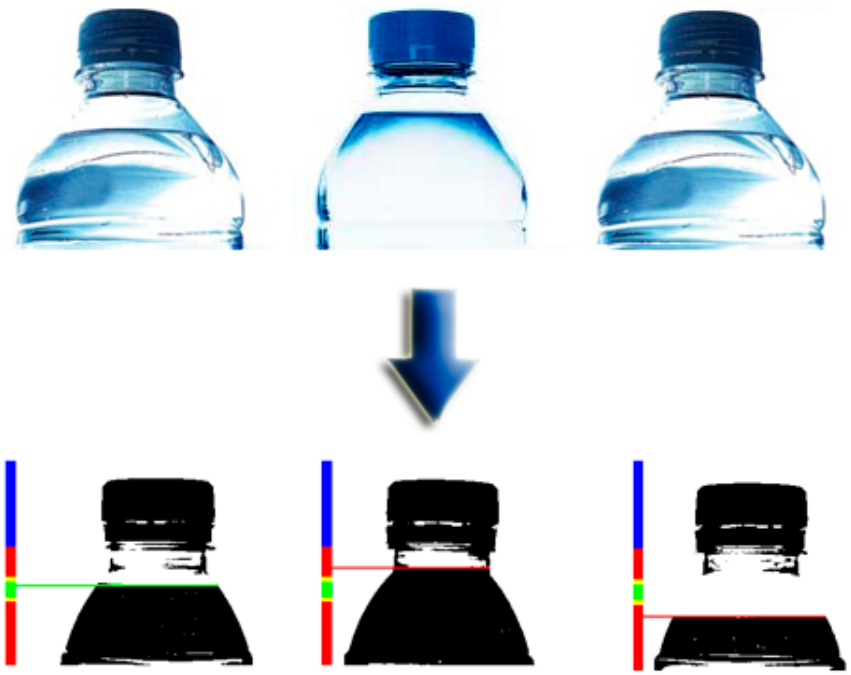


FIGURA 2.13: Proceso de binarizar una imagen.

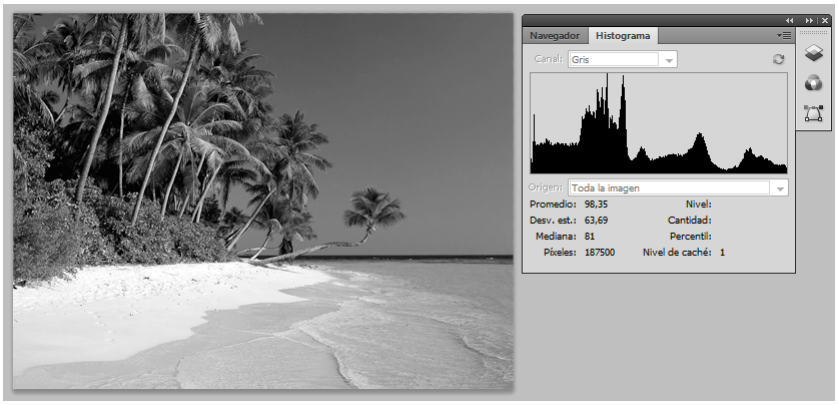



FIGURA 2.14: Histograma en escala de grises.

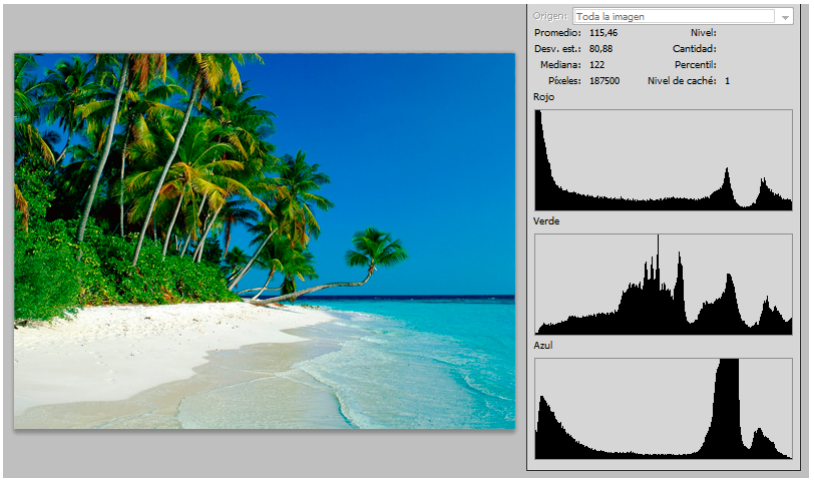



FIGURA 2.15: Histogramas en RGB.

A veces se utilizan combinados el histograma y la binarización porque a través del primero se puede escoger matemáticamente, y por tanto con mayor certeza, el umbral para aplicar después la binarización.

## 2.7. Trabajos similares

En esta sección se presentan tres trabajos que también han implementado un sistema de juego al Tres en Raya para el robot NAO. En los tres el robot se enfrenta a un humano.

El primero de los trabajos ha sido desarrollado por Franck Calzada. El sistema está desarrollado en Python y se ha ayudado del simulador Choregraphe para crear movimientos y de la librería de OpenCV para disponer de un módulo de visión en tiempo real.

El robot NAO es capaz de detectar cuándo hay un tablero situado enfrente de él para poder jugar y cuándo éste ya está completo y no se puede realizar ningún movimiento más. En caso de que no vea totalmente el tablero, comunicará por voz al contrincante que se lo coloque mejor. También comunica por voz cuándo comienza y termina su turno y el del humano y cuándo necesita un rotulador para pintar.

Por otro lado, NAO también analiza el tablero para escoger la siguiente jugada. El símbolo que dibuja NAO sobre cada casilla del tablero es un guión. El siguiente enlace muestra el funcionamiento de este trabajo en un entorno real.

NAO Robot plays TicTacToe with human

El segundo trabajo varía respecto al anterior porque el tablero se encuentra situado en una pared y no sobre una mesa. En este caso, el robot utiliza también visión artificial para identificar la situación del juego y escoger el siguiente movimiento. Por otro lado, ahora en vez de dibujar guiones en el tablero, el robot coloca fichas imantadas. Para ello se las va pidiendo a su contrincante utilizando la voz como canal. El siguiente enlace muestra el funcionamiento de este trabajo en un entorno real.

Nao Robot-playing tic tac toe

El último trabajo es muy similar al anterior en cuanto al entorno del juego, pues también se colocan las equis y los círculos como si fueran piezas en el tablero situado en la pared. Hay una diferencia que sí que cambia respecto a los anteriores trabajos y es que el robot NAO no es autónomo. No se mueve ni dibuja ni coloca, necesita a un ayudante que le vaya colocando el símbolo donde el robot le indica. Sí que lleva implementado un sistema de visión artificial y de

toma de decisiones. El siguiente enlace muestra el funcionamiento de este trabajo en un entorno real.

NAO plays TicTacToe

Comparando cada uno de estos trabajos con el que se quiere desarrollar, se encuentran algunas diferencias. El objetivo de este trabajo es conseguir que NAO dibuje símbolos más complicados que un guión, por ejemplo una equis o un cuadrado. En consecuencia, no se le va a proporcionar ninguna pieza para colocar sobre el tablero. Sobre la colocación del tablero, se deberá analizar qué posición es mejor, si sobre una mesa o colocado en una pared. Principalmente, se quiere conseguir que el grado de autonomía de NAO sea el máximo posible, por lo que tampoco tendrá ningún ayudante que pinte por él ni tampoco que le diga qué casilla ha pintado su rival y cuál debe pintar él después.

# Capítulo 3

# Arquitectura del sistema

En este capítulo se describe el proceso de desarrollo del software que se ha seguido para realizar el trabajo. Se encuentra divido en dos secciones: Análisis y Diseño.

En la sección Análisis se presenta el alcance del trabajo, el catálogo de requisitos y la especificación de casos de uso.

En la sección Diseño se presenta el estudio previo de las alternativas tecnológicas, el entorno operacional y tecnológico definitivo y por último la descripción general y modular de la arquitectura.

## 3.1. Análisis

En esta sección quedan recogidas todas las tareas relativas al análisis del sistema *Tres en Raya*. Primero se enuncia el alcance del trabajo que describe las principales funcionalidades del módulo y establece sus límites. Después se presenta el catálogo de requisitos, imprescindibles para acotar el proyecto y acercarlo más a la fase de diseño. Finalmente se comentan las restricciones de *hardware* y *software* contempladas en el proyecto y los casos de uso para detallar cada una de las interacciones con el usuario que, en este caso, adquiere el rol de contrincante.

### 3.1.1. Alcance del trabajo

El objetivo de esta sección es concretar las principales funcionalidades del sistema y establecer los límites del trabajo para saber qué debe hacer el sistema y qué no.

Se implementará el juego del Tres en Raya para el modo Humano vs Robot. La figura 3.1 resume los componentes que formarán parte del sistema independientemente de las técnicas que se utilicen para desarrollar el juego.

FIGURA 3.1: Componentes del sistema.

Se dispone de un robot NAO H25, una pizarra, dos rotuladores para pizarra, un router inalámbrico y un ordenador para realizar el trabajo.

De lo hablado en las primeras reuniones que se mantuvieron acerca del trabajo, se extrajeron los requisitos de usuario con el objetivo de establecer los límites del trabajo. Son los siguientes:

- El sistema *Tres En Raya* debe ser compatible con el robot NAO H25.
- El robot NAO H25 deberá dibujar con un rotulador dos símbolos diferentes.
- El robot NAO H25 deberá ser capaz de dibujar dichos símbolos en cada casilla del tablero.
- El tablero tendrá tres filas y tres columnas, haciendo un total de nueve casillas.
- Las acciones a implementar deberán ser atómicas, permitiendo la combinación de varias para generar una acción más compleja, siguiendo siempre una estructura jerárquica.
- El robot NAO H25 deberá entender la situación en la que se encuentra el juego en cada momento.
- En cada turno del robot se partirá siempre desde una misma posición inicial tanto propia del robot como del espacio en el que se encuentra.
- El robot deberá decidir en cada turno de forma autónoma qué jugada realizar en función del estado del tablero.

Por otro lado, también se contemplaron algunos aspectos que no incluirá el trabajo:

- El robot no se colocará de forma autónoma en la posición inicial, se perdería mucho tiempo por la poca precisión que tiene el robot en realizar movimientos pequeños.
- El robot no identificará cambios en el símbolo escogido por el usuario, tan sólo se limitará a identificar dónde ha escogido su contrincante pintar dicho símbolo.

### 3.1.2. Catálogo de requisitos

En esta sección se presenta el catálogo de requisitos software tomando como referencia los requisitos de usuario expuestos en el alcance del trabajo. Este catálogo servirá para diseñar el funcionamiento del módulo en la siguiente fase.

Para una mayor organización, éstos están divididos en requisitos del juego, de comunicación, de reconocimiento, de acciones y de entorno de programación y ejecución.

Cada tabla está compuesta de las siguientes celdas:

**ID:** Identificador unívoco del requisito. Todos seguirán la nomenclatura RS-XX donde XX equivale a un número de dos cifras comenzando por 01.

**Título:** Breve descripción del requisito.

**Descripción:** Explicación detallada del requisito

**Prioridad:** Refleja la importancia de que el requisito se encuentre incluido en el módulo. Tomará los valores: alta o baja. Se considera de prioridad alta a un requisito que sea imprescindible para el correcto funcionamiento del sistema. Se considera de prioridad baja a un requisito que no afecte a las funcionalidades básicas del sistema, pues, si no se incluye, el módulo funcionará correctamente.

#### 3.1.2.1. Entorno de programación y ejecución

| | |
|---|---|
| **ID** | RS-01 |
| **Título** | Sistema operativo |
| **Descripción** | El sistema operativo que deberá utilizarse para desarrollar y ejecutar el sistema *Tres en Raya* será Ubuntu 11.10 |
| **Prioridad** | Alta |

TABLA 3.1: Requisito RS-01: Sistema operativo

| | |
|---|---|
| **ID** | RS-02 |
| **Título** | Robot |
| **Descripción** | El robot para el que debe diseñarse el módulo será NAO H25 |
| **Prioridad** | Alta |

TABLA 3.2: Requisito RS-02: Robot

| | |
|---|---|
| **ID** | RS-07 |
| **Título** | Código del módulo |
| **Descripción** | El código del sistema *Tres en Raya* deberá desarrollarse en Python o C++. |
| **Prioridad** | Alta |

TABLA 3.3: Requisito RS-07: Código del módulo

| | |
|---|---|
| **ID** | RS-08 |
| **Título** | NaoQi |
| **Descripción** | La versión del *framework NaoQi* a utilizar será la 1.12.3 o 1.12.5. |
| **Prioridad** | Alta |

TABLA 3.4: Requisito RS-08: NaoQi

| | |
|---|---|
| **ID** | RS-09 |
| **Título** | Choregraphe |
| **Descripción** | La versión del simulador *Choregraphe* a utilizar será la 1.12.3 o 1.12.5. |
| **Prioridad** | Alta |

TABLA 3.5: Requisito RS-09: Choregraphe

| | |
|---|---|
| **ID** | RS-16 |
| **Título** | Comunicación con el ordenador |
| **Descripción** | El robot y el ordenador se comunicarán a través de un modem inalámbrico vía Wi-Fi o Ethernet mediante una red configurada expresamente para dicha comunicación. |
| **Prioridad** | Alta |

TABLA 3.6: Requisito RS-16: Comunicación con el ordenador

#### 3.1.2.2. Acciones

| | |
|---|---|
| **ID** | RS-03 |
| **Título** | Dibujar |
| **Descripción** | El robot deberá aprender a dibujar utilizando la técnica de aprendizaje por demostración. |
| **Prioridad** | Alta |

TABLA 3.7: Requisito RS-03: Dibujar

| | |
|---|---|
| **ID** | RS-11 |
| **Título** | Dibujar símbolos |
| **Descripción** | El robot deberá ser capaz de dibujar dos símbolos distintos en cada casilla del tablero. |
| **Prioridad** | Alta |

TABLA 3.8: Requisito RS-11: Dibujar símbolos

| | |
|---|---|
| **ID** | RS-12 |
| **Título** | Acciones atómicas |
| **Descripción** | Se implementarán acciones atómicas siempre que sea posible. |
| **Prioridad** | Alta |

TABLA 3.9: Requisito RS-12: Acciones atómicas

| | |
|---|---|
| **ID** | RS-13 |
| **Título** | Acciones complejas |
| **Descripción** | Se combinarán acciones atómicas para construir acciones más complejas siempre siguiendo una jerarquía. |
| **Prioridad** | Alta |

TABLA 3.10: Requisito RS-13: Acciones complejas

| | |
|---|---|
| **ID** | RS-20 |
| **Título** | Actuación ante caídas |
| **Descripción** | Se deberán incorporar acciones que permitan al robot levantarse y sentarse para recuperar la normalidad en el juego en caso de sufrir una caída. |
| **Prioridad** | Baja |

TABLA 3.11: Requisito RS-20: Actuación ante caídas

| | |
|---|---|
| **ID** | RS-19 |
| **Título** | Posición inicial |
| **Descripción** | El robot se colocará en una posición tal que podrá pintar en la primera fila tercera columna sin tener que realizar ningún tipo de desplazamiento. |
| **Prioridad** | Alta |

TABLA 3.12: Requisito RS-19: Posición inicial

#### 3.1.2.3. Juego

| | |
|---|---|
| **ID** | RS-04 |
| **Título** | Tablero |
| **Descripción** | El tablero deberá tener nueve casillas repartidas en tres filas y tres columnas. |
| **Prioridad** | Alta |

TABLA 3.13: Requisito RS-04: Tablero

| | |
|---|---|
| **ID** | RS-05 |
| **Título** | Medidas de las casillas |
| **Descripción** | La altura y anchura de cada casilla debe ser la misma en todos los casos. |
| **Prioridad** | Alta |

TABLA 3.14: Requisito RS-05: Casilla

| | |
|---|---|
| **ID** | RS-06 |
| **Título** | Numeración de las casillas |
| **Descripción** | Cada casilla deberá numerarse de forma unívoca. |
| **Prioridad** | Alta |

TABLA 3.15: Requisito RS-06: Numeración de las casillas

| | |
|---|---|
| **ID** | RS-15 |
| **Título** | Comunicación con el contrincante |
| **Descripción** | Se deberá comunicar al contrincante de forma escrita o por voz cuando puede comenzar su jugada y cuándo comienza la del robot. |
| **Prioridad** | Alta |

TABLA 3.16: Requisito RS-15: Comunicación con el contrincante

| | |
|---|---|
| **ID** | RS-17 |
| **Título** | Duración del juego |
| **Descripción** | El juego no debe superar los 10 minutos de duración. |
| **Prioridad** | Alta |

TABLA 3.17: Requisito RS-17: Duración del juego

| | |
|---|---|
| **ID** | RS-18 |
| **Título** | Fin del juego |
| **Descripción** | El robot debe comunicar al contrincante si ha ganado o ha perdido el juego. |
| **Prioridad** | Alta |

TABLA 3.18: Requisito RS-18: Fin del juego

#### 3.1.2.4. Reconocimiento

| | |
|---|---|
| **ID** | RS-10 |
| **Título** | Toma de decisiones |
| **Descripción** | El robot deberá escoger la siguiente jugada mediante búsqueda heurística. |
| **Prioridad** | Alta |

TABLA 3.19: Requisito RS-10: Toma de decisiones

| | |
|---|---|
| **ID** | RS-14 |
| **Título** | Reconocer jugada |
| **Descripción** | El robot será capaz de reconocer la jugada que ha realizado su contrincante. |
| **Prioridad** | Alta |

TABLA 3.20: Requisito RS-14: Reconocer jugada

| | |
|---|---|
| **ID** | RS-19 |
| **Título** | Validación humana |
| **Descripción** | Un humano confirmará en cada turno, a través del sistema, que el reconocimiento de la jugada del contrincante realizado por el robot es correcto. Si el robot no ha acertado deberá corregirle. |
| **Prioridad** | Alta |

TABLA 3.21: Requisito RS-19: Validación humana

### 3.1.3. Restricciones Hardware

- Se dispone de un robot Nao H25 que, por cuestiones ajenas al proyecto, tiene inhabilitada la mano izquierda.
- El robot Nao H25 tiene dos cámaras, una situada en la frente y otra en la boca. No pueden ser utilizadas simultáneamente.
- El robot tiene una autonomía de 40 minutos cuando se ejecutan acciones simultáneas como visión y desplazamiento.
- El ordenador deberá disponer como mínimo de 1.5 GHz CPU, 1 GB RAM, tarjeta de red habilitada para Ethernet/Wi-Fi y una tarjeta gráfica certificada con OpenGL.

### 3.1.4. Restricciones Software

- El sistema operativo instalado en el ordenador deberá ser Linux, distribución Ubuntu 11.04 de 32 bits.
- Debe estar instalado el software de Nao en el ordenador desde el que se ejecutará el módulo: Choregraphe y NaoQi en las versiones 1.12.3.
- El idioma con el que se comunica por voz el robot Nao es inglés.

### 3.1.5. Especificación de Casos de uso

El principal objetivo de una especificación de casos de uso es obtener una primera idea de las funcionalidades que tendrá el sistema a implementar. Estos casos de uso han sido extraídos de la especificación de requisitos de software recogida en el apartado anterior.

Los casos de uso reflejan la interacción del usuario con el sistema, en este caso, el jugador humano como contrincante en el sistema *Tres en Raya* equivale al actor del que se habla en los posteriores casos de uso.

Cada caso de uso está compuesto de la siguiente información:

**ID:** Identificador unívoco del caso de uso. Todos seguirán la nomenclatura CU-XX donde XX equivale a un número de dos cifras comenzando por 01.

**Actor:** Usuario que interactúa con el sistema

**Título:** Breve descripción del caso de uso

***Trigger*:** Desencadenante que provoca que suceda el caso de uso

**Secuencia de pasos:** Desarrollo del escenario del caso de uso

**Secuencias alternativas o de error:** Desarrollo del escenario cuando se ha producido algo inesperado

A continuación se describen todos los casos de uso identificados para el sistema *Tres en Raya*.

| | |
|---|---|
| **ID** | CU-01 |
| **Actor** | Contrincante |
| **Título** | Iniciar juego |
| ***Trigger*** | El jugador humano (contrincante) quiere jugar al Tres en Raya con el robot Nao. |
| **Secuencia de pasos** | 1. El jugador humano pone en marcha el módulo |
| **Secuencia alternativa o de error** | 1. El contrincante se arrepiente y no comienza el juego<br>2. El contrincante abandona el juego antes de realizar la primera jugada |

TABLA 3.22: Caso de Uso CU-01: Iniciar juego

| | |
|---|---|
| **ID** | CU-02 |
| **Actor** | Contrincante |
| **Título** | Iniciar jugada |
| ***Trigger*** | El robot ha terminado su turno |
| **Secuencia de pasos** | 1. El contrincante evalúa el estado del juego<br>2. El contrincante escoge una casilla vacía y la marca con su símbolo |
| **Secuencia alternativa o de error** | 1. El contrincante se equivoca de casilla y pinta en una ya marcada<br>2. El contrincante abandona el juego |

TABLA 3.23: Caso de Uso CU-02: Iniciar jugada

| | |
|---|---|
| **ID** | CU-03 |
| **Actor** | Contrincante o persona ajena al juego |
| **Título** | Confirmación del análisis del tablero |
| ***Trigger*** | El robot ha analizado el tablero |
| **Secuencia de pasos** | 1. El sistema pide al actor que confirme si el robot ha realizado con éxito el reconocimiento de la jugada del contrincante<br>2. El actor comprueba si la casilla detectada es la correcta<br>3. El actor da una respuesta afirmativa al sistema |
| **Secuencia alternativa o de error** | 1. El actor da una respuesta negativa al sistema<br>2. El sistema le pide que corrija el error<br>3. El actor proporciona la información correcta al sistema |

TABLA 3.24: Caso de Uso CU-03: Confirmación del análisis del tablero

El diagrama de casos de uso queda recogido en la figura 3.2.

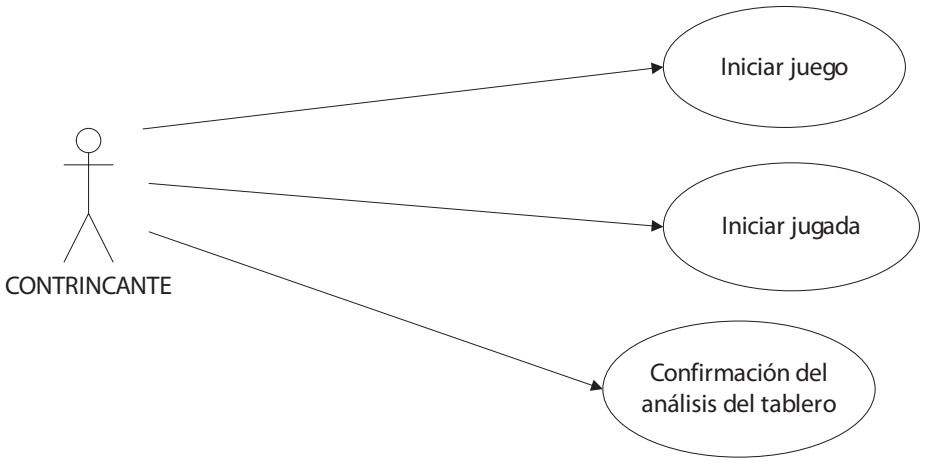



FIGURA 3.2: Diagrama de casos de uso.

## 3.2. Diseño

En esta sección se describe el diseño de la arquitectura a implementar. Primero se hace un pequeño estudio de las alternativas disponibles, después se habla de la toma de decisiones y la justificación de cada una de ellas y finalmente se describe el sistema tanto general como modular para explicar con detalle cada componente.

### 3.2.1. Estudio del entorno operacional y tecnológico

El sistema *Tres en Raya* está subdividido en dos partes clave: dibujo de símbolos y análisis del tablero. Para desarrollar cada parte se presentan diferentes alternativas.

Como el dibujo de símbolos se va a realizar utilizando aprendizaje por demostración, las opciones existentes para grabar las instancias de ejemplo son las siguientes: Teleoperación utilizando Kinect, Teleoperación utilizando un dispositivo Android y generación de comportamientos con Choregraphe en el modo animación.

Por otro lado, el análisis del tablero se realiza en cada turno del robot mediante visión artificial. Las alternativas a estudiar son utilizar los histogramas para reconocer cambios en la cantidad de color de una casilla o reconocer formas que permitan distinguir entre una casilla vacía y otra con un símbolo.

Adicionalmente, se estudiaron las posibles situaciones del tablero frente a la del robot, pues era importante decidir si el robot se desplazaría o no para pintar sobre el tablero y si éste se

encuentra en posición paralela o perpendicular al robot ya que los movimientos para pintar en los dos casos son diferentes.

#### 3.2.1.1. Estudio de la grabación de instancias

Como ya se ha anunciado previamente, se evaluaron tres alternativas diferentes: Kinect, Android y Choregraphe.

Para realizar la grabación de instancias mediante Kinect se reutilizó el código desarrollado en [17]. A la hora de ejecutar el código, la cámara Kinect calibraba al usuario que iba a realizar la teleoperación en cuanto éste decía "*start*". Una vez hecho esto ya podía moverse para que NAO imitase sus movimientos. Para grabar instancias de ejemplo había que obtener el estado de las articulaciones y la información de los sensores en cada instante de tiempo, una tarea fácil teniendo en cuenta que el código ya incluía esos datos. Cuando el usuario terminase el comportamiento a reproducir tan sólo tenía que decir "*finish*" y la ejecución se detendría inmediatamente. La figura 3.3 muestra un ejemplo de teleoperación utilizando dicho sistema.

FIGURA 3.3: Ejemplo de teleoperación con la cámara Kinect y NAO.

El principal problema que surgió durante la teleoperación fue la poca precisión que existía en la traducción de los movimientos de la muñeca del humano al robot para que éste último consiguiera imitar el comportamiento de dibujar. Así era muy difícil desarrollar esta tarea. Por otro lado, como el sistema tenía reconocimiento por voz, en algún caso debido al ruido del ambiente detenía la ejecución.

El sistema de teleoperación con Android era muy similar al de Kinect, sólo que en vez de moverse directamente el usuario, los desplazamientos de las articulaciones del robot se realizaban a través de la interfaz de la pantalla del teléfono. Ésta contenía dos *joystick* con varias opciones, en función de la articulación seleccionada. Para iniciar la teleoperación tan sólo había que iniciar en el teléfono la aplicación NaoController. La figura 3.4 muestra un ejemplo de teleoperación utilizando dicho sistema.

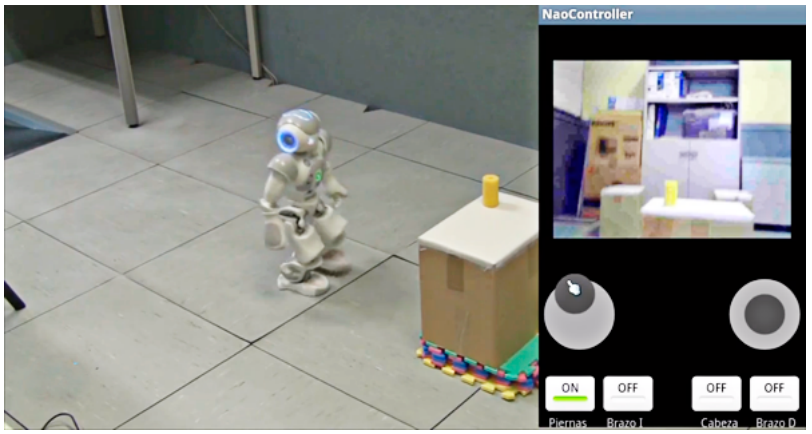



FIGURA 3.4: Ejemplo de teleoperación con el dispositivo Android y NAO.

El problema que surgía con esta alternativa es que era muy complicado mover más de dos articulaciones simultáneamente porque había que acceder primero a un menú para seleccionar la deseada. Esto hacía que la tarea de dibujar fuese bastante lenta y costosa de reproducir. También se necesitaba adquirir mucha práctica para no mover demasiado rápido o lento los *joysticks*. Respecto a la grabación de los movimientos, no habría problema alguno en almacenar los valores porque también los recogía el código.

Respecto a la última alternativa, la tarea de dibujar se realiza de forma diferente porque no es necesaria la teleoperación. Dentro de Choregraphe existe una opción llamada modo animación en la que se desbloquean las articulaciones deseadas y según mueve el usuario esa parte del robot se van grabando los valores de las articulaciones en paralelo con el tiempo de desplazamiento de cada una hasta que el comportamiento termina. La principal desventaja de esta solución es que inicialmente no es fácil mover al robot para obtener un comportamiento que después reproduzca con éxito. Cuando el usuario graba un comportamiento influye cualquier pequeño empujón. También hay que darse cuenta de si verdaderamente se está desplazando la articulación o se la está forzando, pues si sucede esto último no se grabará nada en el comportamiento a pesar de que aparente que sí.

De todas estas alternativas, por la rapidez con la que se obtienen las instancias y porque se trabaja directamente sobre el robot, se escogió la de Choregraphe y el modo animación. De esta forma evitamos trabajar a distancia y los comportamientos se aligeran en cuanto a velocidad. Además todo el comportamiento grabado se exporta directamente desde Choregraphe a Python o C++, lo que facilita enormemente la ejecución posterior en el robot.

#### 3.2.1.2. Estudio del análisis del tablero

En este caso se estudian dos alternativas: aplicar histogramas con o sin binarización o aplicar reconocimiento de formas.

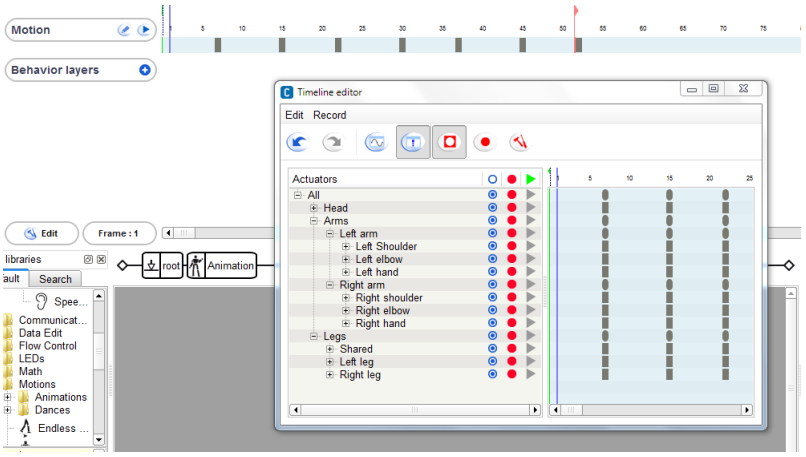



FIGURA 3.5: Modo animación con Choregraphe. (Fuente)

Para aplicar la alternativa de los histogramas era necesario tomar una foto de cada casilla vacía en cada turno del robot, lo que podría entenderse como una desventaja ya que ralentiza el análisis del tablero. Por otro lado, la similitud de dos histogramas de una misma casilla en distintos instantes de tiempo es muy fácil de calcular por métodos estadísticos, obteniendo una tasa de acierto alta. Si los dos histogramas varían mucho significará que se ha pintado un símbolo en la casilla correspondiente. Por otro lado, si se le añade el proceso de binarizado, nos enfrentamos a dos situaciones: que la binarización ayude porque marcaría en negro el símbolo y en blanco la casilla vacía, lo que aumentaría la tasa de acierto de detección de cambios en la imagen o que como la foto es un poco más grande que la casilla, convierta el fondo a un tono grisáceo cercano al negro haciendo que aumenten los valores de la izquierda en el histograma y confundan al sistema engrandeciendo la probabilidad de que la casilla se detecte como pintada.

Para implementar esta primera alternativa en el sistema se estudiaron distintas librerías de programación de las cuáles las más utilizadas y recomendadas eran OpenCV y Python Imaging Library (PIL). En la figura 3.6 se puede ver el tipo de histograma que se obtiene con OpenCV.

En relación con la segunda alternativa, reconocimiento de formas, se estudió utilizar el simulador Choregraphe porque incluye una herramienta para que el robot sea capaz de reconocer objetos en función de su forma y color. Esto facilita mucho la transmisión del conocimiento de las formas al robot. El proceso a seguir es el siguiente: dentro de Choregraphe, se congelaría la imagen en la que se encuentra el objeto, después se va marcando sobre la imagen la silueta del objeto y se le pide al simulador que lo guarde en el NaoQi del robot. Esa silueta lleva asignados un nombre y una orientación. Esto último se guarda por si se desea capturar la silueta del mismo objeto desde diferentes ángulos. En la figura 3.7 se puede ver gráficamente parte del proceso.

Aplicando esto al sistema *Tres en Raya* habría que enfocar a cada una de las casillas y el sistema identificaría cuáles han sido elegidas por el robot o por el humano. El principal problema entonces está relacionado con esto último. Si se aplicase esta alternativa habría que configurar el

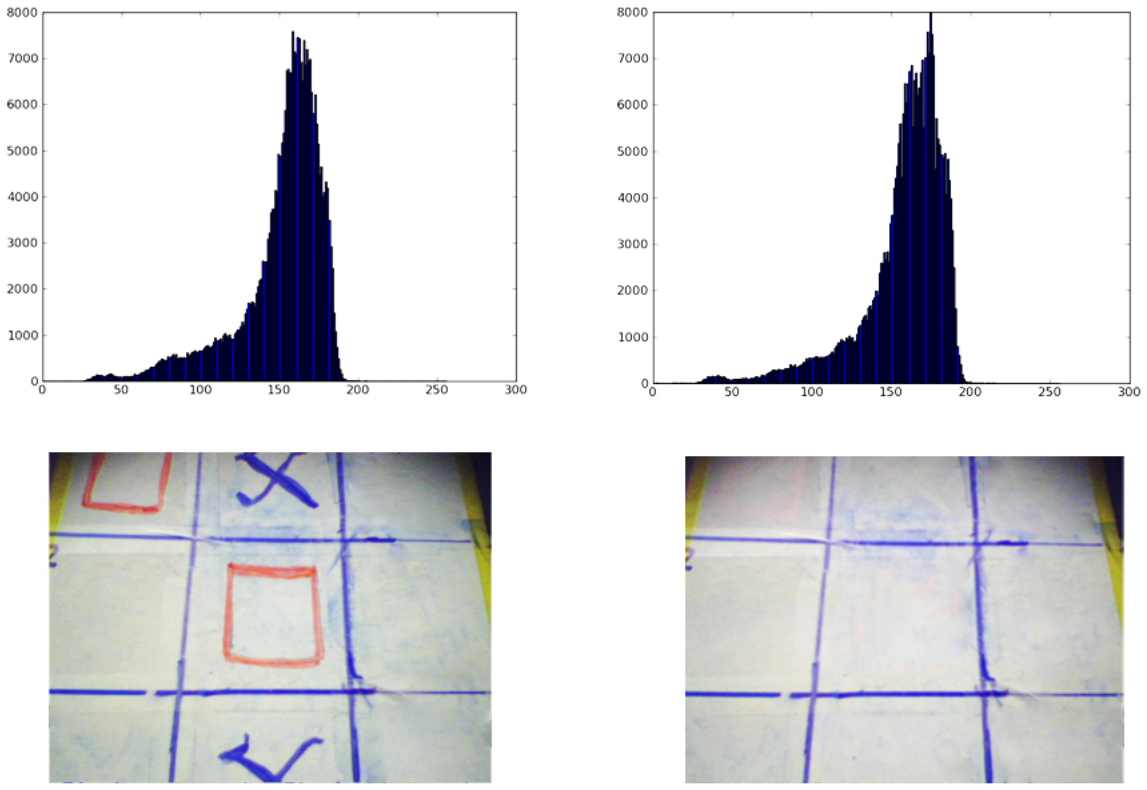



FIGURA 3.6: Histogramas obtenidos con OpenCV.

juego siempre antes de empezar a jugar porque el símbolo del contrincante puede ser diferente a los utilizados por otros.

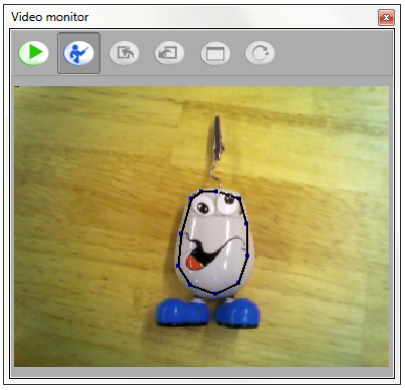



FIGURA 3.7: Reconocimiento de objetos con Choregraphe. (Fuente)

De las tres alternativas presentadas, se escogió la alternativa de los histogramas porque con la alternativa del reconocimiento de formas es obligatorio configurar el juego cada vez que se quiere iniciar una partida, lo que implica disponer de la licencia del software Choregraphe y de un experto que sepa cómo capturar cada uno de los símbolos y transmitírselos al robot. Teniendo en cuenta que las partidas tardarán entre 5-10 minutos en resolverse, es una gran desventaja

realizar toda esa configuración inicial por el coste y el tiempo que le supone al sistema. Sobre la técnica del binarizado, no se estimó necesario incluirla porque como no se puede apreciar durante el juego cómo transforma dicho proceso las imágenes, utilizarla podría empeorar la tasa de acierto obtenida con la alternativa de los histogramas.

#### 3.2.1.3. Estudio de la situación del tablero frente al robot NAO

A la hora de pensar en cómo grabar las instancias referentes al dibujo de símbolos, se plantearon dos entornos diferentes:

1. El robot NAO se encuentra frente al tablero, situado en una pared, y para pintar se desplaza a izquierda y derecha.
2. El tablero se sitúa encima de una mesa, caja o apoyo similar y el robot pinta desde arriba y sin desplazarse.

El principal objetivo que debían permitir estos dos modelos era que el robot alcanzara las nueve casillas del tablero sin dificultad. En el segundo modelo esto era más complicado de cumplir porque la casilla central solía quedarse en un ángulo muerto para el robot. Además la ventaja de utilizar el primer modelo, a pesar de que el robot tuviera que desplazarse, es que puede adaptarse de forma sencilla a las dimensiones de un nuevo tablero. Por este motivo se escogió la configuración número uno.

### 3.2.2. Entorno tecnológico y operacional

En este apartado se describen los elementos de los que está compuesto el sistema *Tres en Raya*. Es imprescindible que cada uno de ellos se encuentre presente en el entorno si éste quiere reproducirse.

- Un robot Nao H25 con los comportamientos *Stand* y *SitDown* incluídos.
- Un router inalámbrico.
- Un ordenador con 3 GHz y 4 GB de RAM con sistema operativo Ubuntu 11.04.
- Choregraphe 1.12.3 instalado en el ordenador.
- NaoQi 1.12.3 instalado en el ordenador.
- Librerías de Python (en principio instaladas por defecto por Ubuntu 11.04).
- Librería OpenCV.

- Librería AIMA.
- Pizarra estilo Vileda.
- Dos rotuladores de colores diferentes para pizarra.
- Folio DIN-A4 con la referencia de Nao frente a la pared.

Los siguientes subapartados están dedicados a presentar brevemente algunos de los componentes mencionados anteriormente para profundizar un poco más en las capacidades que ofrece cada uno al sistema.

#### 3.2.2.1. El robot NAO

El robot humanoide NAO H25 dispone de múltiples sensores (micrófonos, cámaras, giroscopio, acelerómetro, etc.) y actuadores (altavoces, motores, etc.). Tiene 25 grados de libertad, por lo que posee una amplia movilidad. Sus creadores son la empresa francesa, Aldebaran Robotics [18]. Tal es el éxito que ha obtenido este robot a nivel internacional que se ha convertido, entre otras cosas, en el robot más utilizado de la Robocup [19][20].

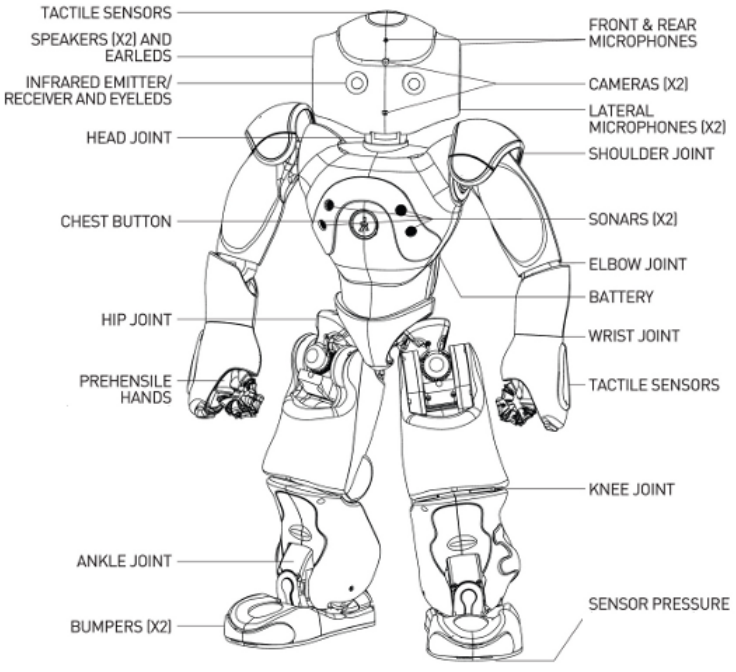



FIGURA 3.8: Croquis Robot Nao H25. (Fuente)

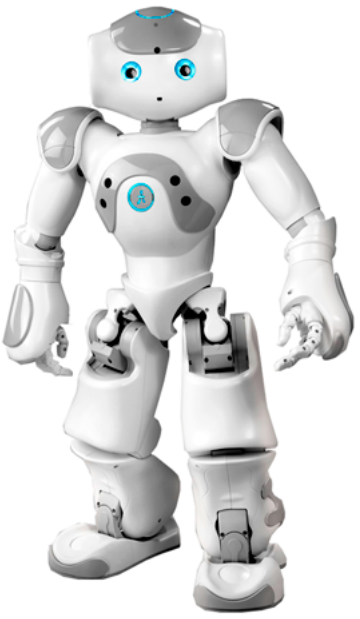


FIGURA 3.9: Robot Nao H25. (Fuente)

El control de Nao desde el punto de vista de sus articulaciones se divide en ocho partes diferentes. A continuación se describe brevemente lo que es capaz de hacer cada una de ellas. Si se necesita un mayor nivel de detalle se puede consultar la referencia a pie de página [1].

**Head:** La cabeza del robot puede moverse de izquierda a derecha (*yaw*) y de arriba a abajo (*pitch*).

**LArm:** El *Left arm* o brazo izquierdo permite mover el hombro (*shoulder*) sobre sí mismo (*roll*) o de arriba a abajo (*pitch*) y el codo (*elbow*) sobre sí mismo (*roll*) o de izquierda a derecha (*yaw*).

**LHand:** La *Left hand* o mano izquierda permite abrir y cerrar los dedos de la mano (*LHand*) y girar la muñeca sobre sí misma (*LWristYaw*).

**LLeg:** La *Left leg* o pierna izquierda permite subir y bajar ambas piernas (*LHipYawPitch*), girar y mover la pierna izquierda (*LHipRoll, LHipPitch*), mover la rodilla de arriba a abajo (*LKneePitch*) y mover y girar el tobillo (*Ankle*).

**RArm:** El *Right arm* o brazo derecho permite mover el hombro (*shoulder*) sobre sí mismo (*roll*) o de arriba a abajo (*pitch*) y el codo (*elbow*) sobre sí mismo (*roll*) o de izquierda a derecha (*yaw*).

**RHand:** La *Right hand* o mano derecha permite abrir y cerrar los dedos de la mano (*RHand*) y girar la muñeca sobre sí misma (*RWristYaw*).

**RLeg:** La *Right leg* o pierna derecha permite girar y mover la pierna derecha (RHipRoll, RHipPitch), mover la rodilla de arriba a abajo (*RKneePitch*) y mover y girar el tobillo (ankle). El movimiento de *RHipYawPitch* es el mismo que el de *LHipYawPitch*, pues se mueven las dos piernas simultáneamente. Por lo tanto, comparten el mismo valor y podemos utilizar indistintamente uno u otro.

**Torso:** El torso alberga los cuatro sónar, el giroscopio y el acelerómetro del robot. Estos elementos le permiten ser capaz de prevenir el choque con obstáculos o equilibrarse entre otros. En la imagen se señalan los cuatro sónar, siendo dos emisores (*transmitter*) y dos receptores (*receiver*). Los primeros lanzan el sónar y los segundos recogen los datos obtenidos de a cuánta distancia se encuentra el objeto más cercano.

#### 3.2.2.2. El simulador Choregraphe

Este simulador ha sido desarrollado por Aldebaran Robotics con el fin de facilitar al usuario la forma de trabajar con los robots Nao. En Choregraphe [21] se pueden diseñar y ejecutar distintos

[1] https://developer.aldebaran-robotics.com/doc/1-12/nao/hardware/kinematics/nao-joints-33.html

comportamientos, probarlos sobre un robot virtual o sobre el real, acceder a la cámara del robot, etc. Es una herramienta fundamental para conocer el estado del robot y trabajar independientemente sobre cada una de sus articulaciones.

El punto fuerte de Choregraphe es su interfaz, sencilla e intuitiva. Para que el robot del simulador realice un movimiento podemos combinar los comportamientos ya desarrollados del menú de la izquierda o crear otros nuevos mediante la opción de "animación". Para ejecutarlos simplemente hay que pulsar el botón Run. Al trabajar siempre sobre diferentes ventanas todo está muy bien ordenado y la flexibilidad que ofrece es inmensa. A través del menú de la parte superior se puede acceder a todo el resto de información que esta herramienta proporciona. Para aprender a trabajar con Choregraphe, Aldebaran Robotics elaboró una guía de usuario con diferentes tutoriales de dificultad creciente [2].

#### 3.2.2.3. NaoQi

NaoQi [22] es el *framework* desarrollado por Aldebaran Robotics que sirve de puente entre la máquina y NAO, controlando la ejecución de cualquier comportamiento y capturando la información de todos los sensores con el único objetivo de que todo se produzca con éxito y sin incidentes. Este *framework* se encuentra dentro del robot, a modo de cerebro, y también en el ordenador desde el que se envían las órdenes.

El framework se encuentra dividido en seis grandes áreas diferenciadas claramente por su funcionalidad. Son las siguientes:

- NAOqi Core: contiene un conjunto de módulos relacionados con los comportamientos genéricos del robot, la memoria del robot y los ficheros de configuración.
- NAOqi Motion: contiene todas las funciones que permiten al robot desplazarse, controlando el equilibrio, el estado de las articulaciones y la prevención de choques o caídas.
- NAOqi Audio: contiene los módulos que le permiten al robot interactuar con el usuario mediante el habla, la reproducción de sonidos y el reconocimiento de voz.
- NAOqi Vision: contiene los módulos que permiten grabar desde las cámaras del robot y realizar reconocimiento de caras y de objetos.
- NAOqi Sensors: contiene los módulos que proporcionan información sobre el estado del robot: sensores, batería, pose en la que se encuentra etc.

[2] http://www.aldebaran-robotics.com/documentation/software/choregraphe/index.html

- NAOqi Trackers: contiene las funciones que permiten seguir a una persona a través de reconocimiento facial, a un objeto o a una marca.

A través del API de NaoQi se pueden desarrollar nuevos módulos en Python, C++ o Lisp en vez de utilizar la opción de Choregraphe. De esta forma, se puede conseguir una mayor flexibilidad y personalización de éstos y además será posible adentrarse mucho más en el robot.

Es habitual encontrarse códigos que combinan el API de NaoQi no sólo con las librerías de Python o C++, sino incluso con paquetes de ROS (Robot Operating System) [23].

#### 3.2.2.4. OpenCV

OpenCV (Open Source Computer Vision) [24] es una librería de programación desarrollada para proporcionar algoritmos y funciones que permitan trabajar en tiempo real con visión artificial. Fue desarrollada inicialmente por Intel. Después varios de sus creadores terminaron en Willow Garage y Itseez donde siguieron desarrollando con gran éxito la librería acercándola al mundo de la robótica y de los videojuegos. Esta librería se convirtió en un gran referente en el área de la robótica porque se incluyó en el coche Stanley, ganador del DARPA Challenge de 2005. Tiene implementaciones en C++, Python y funciona en sistemas operativos Windows, Mac, Linux, iOS y Android. Actualmente se trabaja en una implementación en JAVA y en una interfaz para CUDA.

Las áreas de trabajo en las que se puede aplicar OpenCV son muchísimas. Centrándonos exclusivamente en el soporte de OpenCV para la robótica, las principales funcionalidades que se pueden implementar en un robot se recogen en la figura 3.10.

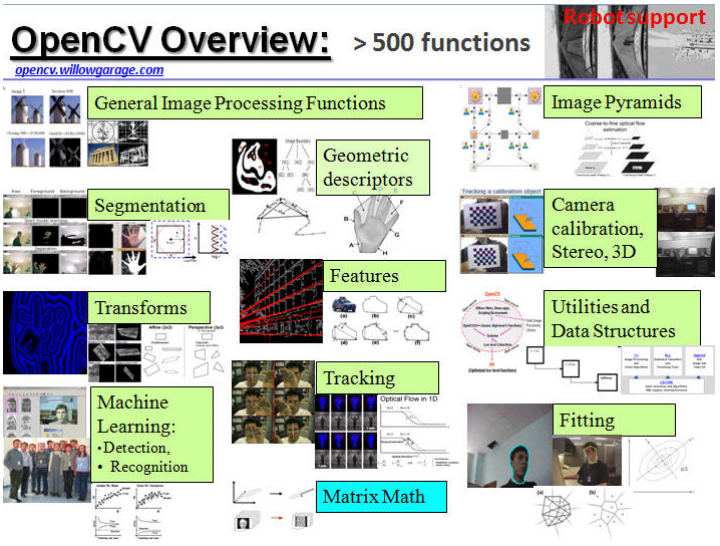



FIGURA 3.10: Áreas de trabajo de OpenCV en robótica. (Fuente)

#### 3.2.2.5. AIMA

La librería AIMA fue desarrollada para el libro Artificial Intelligence: A Modern Approach [25]. En la actualidad, dicho libro es el estado del arte de la inteligencia artificial, entre otras cosas, por ser la segunda publicación más citada en el siglo actual [3].

AIMA está implementada en Lisp, Python y Java y cuenta con numerosas funciones que te permiten trabajar con muchas técnicas relacionadas con la inteligencia artificial. Algunos ejemplos son: aprendizaje automático, búsqueda heurística, probabilidad y agentes.

Para utilizar la librería AIMA tan sólo hay que descargarse los ficheros con las funciones que se quieran incluir en el código, importar los ficheros y llamar a las funciones.

### 3.2.3. Descripción general

En esta sección se va a presentar de forma global la arquitectura diseñada. Para ello primero hay que destacar que la arquitectura se encuentra dividida en tres módulos diferenciados: módulo de dibujo, módulo de visión y módulo del algoritmo del juego. Cada uno de ellos tiene varias funcionalidades que son imprescindibles para el correcto funcionamiento del sistema, pero esto se explicará con mayor detalle en la descripción modular.

La figura 3.11 refleja la comunicación entre los distintos módulos que presenta la arquitectura. La salida de un módulo es la entrada del siguiente. El diagrama se ha realizado desde el punto de vista del robot, pues el jugador humano tan sólo tiene que escoger la siguiente jugada y no interviene en ningún otro proceso.

Tomando como origen el fin de turno del jugador humano, el robot procederá a analizar cada casilla del tablero gracias a las capturas que va tomando de cada una de ellas con sus cámaras. Cuando haya terminado de capturar las imágenes, las comparará con las del turno anterior. Para ello, extrae el histograma de cada par de imágenes y aplica la prueba de Chi-cuadrado. El mayor valor obtenido de todos los pares de imágenes equivale a la jugada que ha realizado el contrincante. Una vez el sistema dispone de esa información, el algoritmo Minimax junto con la poda Alfa-beta exploran las diferentes jugadas disponibles en el juego e indican la más ventajosa para el robot. Ésta será la que el robot dibuje gracias a la jerarquía de acciones desarrollada en el módulo de dibujo.

Para entender mejor el funcionamiento del juego se presenta en la figura 3.12 el diagrama de flujo general de la arquitectura.

[3] http://aima.cs.berkeley.edu/index.html

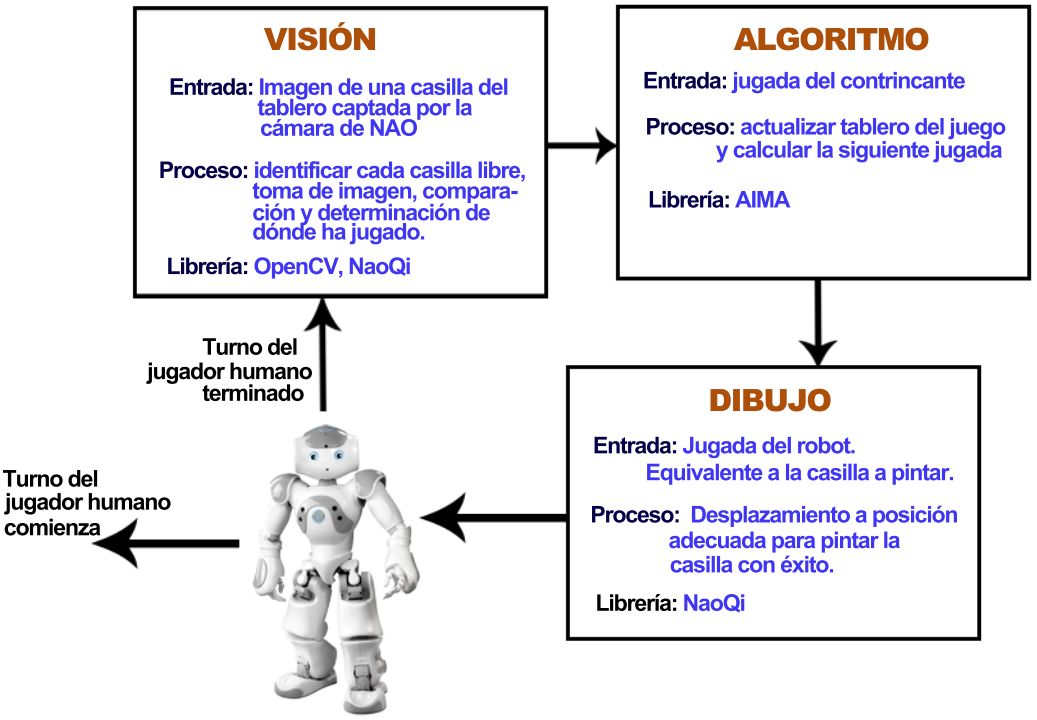



FIGURA 3.11: Estructura global de la arquitectura Tres en Raya.

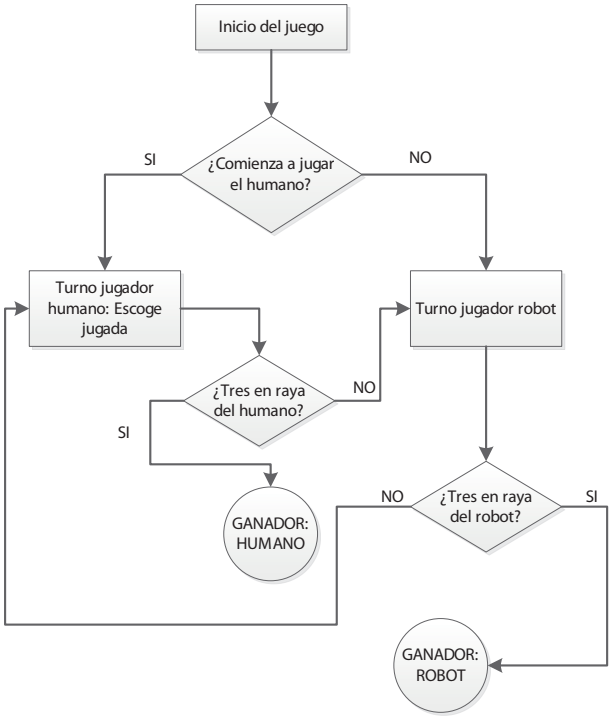



FIGURA 3.12: Diagrama de flujo de la arquitectura Tres en Raya.

El juego comienza escogiendo de forma aleatoria el jugador que empieza a pintar en el tablero. Tras el turno del humano y del robot o viceversa, se comprueba si se ha obtenido alguna jugada

ganadora. Si esto es cierto, el juego se detendrá y anunciará el ganador.

Como se puede observar, el desarrollo del turno del robot no es tan sencillo de realizar como el del humano. Por este motivo, se muestra en la figura 3.13 un diagrama de flujo dedicado exclusivamente al desarrollo del turno del robot.

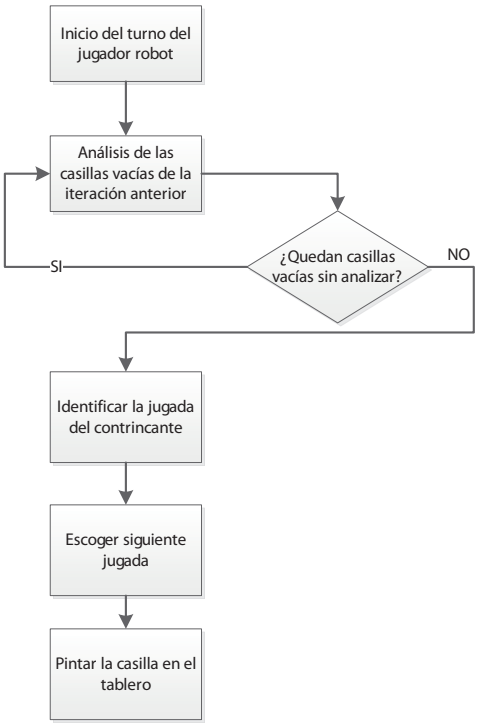



FIGURA 3.13: Diagrama de flujo del turno del robot.

### 3.2.4. Descripción modular

En esta sección se presentan detalladamente cada uno de los módulos que componen la arquitectura del sistema. El orden establecido para explicar cada uno de ellos es el que se siguió en el desarrollo del sistema.

Durante la explicación se mencionará en numerosas ocasiones la palabra primitiva, cuyo significado es aquella acción de es atómica y no puede dividirse en otras más pequeñas.

#### 3.2.4.1. Módulo de dibujo

En este módulo se recogen todas las acciones que permiten a NAO dibujar un símbolo en cualquier casilla del tablero. Dichas acciones están divididas en poses, desplazamientos y dibujo de símbolos y siempre que ha sido posible se han desarrollado siguiendo una estructura jerárquica.

Las que se encuentran en el nivel más bajo, sin descendientes, son las llamadas primitivas, comportamientos o acciones que no se pueden simplificar más. Según se va ascendiendo en el árbol se encuentran acciones complejas formadas por un conjunto de primitivas.

La figura 3.14 recoge las primitivas relacionadas con comportamientos que el robot ya sabía realizar antes de programar este sistema. La primitiva *Stand-Up* permite al robot levantarse desde el suelo. Por el contrario, la primitiva *Sit-Down* permite al robot sentarse en el suelo. La primitiva *Fall-Down Recovery* permite al robot recuperarse de una caída cuando detecta que ésta se ha producido. Esto es posible porque el robot incluye un sensor dedicado exclusivamente a esa tarea. Finalmente, la primitiva *pose Init* prepara al robot en la pose adecuada para comenzar a dibujar o a desplazarse en cualquier dirección eliminando cualquier riesgo de caída o desequilibrio. Ésta viene ya predefinida en el robot así que sólo hay que activarla cuando se quiera utilizar. NAO flexionará levemente las rodillas, inclinará su cuerpo hacia adelante y también dejará los brazos a ambos lados levemente flexionados. Se escogió esta pose porque es con la que mejor se trabaja para realizar movimientos de agacharse, levantarse y dibujar. La figura 3.15 muestra al robot NAO en la pose INIT.


FIGURA 3.14: Estructura de acciones para para poses y comportamientos.

FIGURA 3.15: Pose INIT del robot.

Las figuras 3.16, 3.17 y 3.18 recogen las primitivas relacionadas con los desplazamientos del robot. Éste puede moverse en las cuatro direcciones: izquierda, derecha, delante y detrás, también podrá agacharse flexionando las rodillas y levantarse. Como se puede apreciar en las diferentes figuras, dependiendo de la fila en la que se encuentre la casilla, el robot empleará un conjunto más amplio o más reducido de primitivas.

La acción situada en el nodo raíz se ejecuta siguiendo el orden de los nodos hoja de izquierda a derecha. Para poder dibujar en las casillas de la primera fila, el robot sólo tendrá que desplazarse a la izquierda (figura 3.16). Por otro lado, para pintar en la segunda fila será necesario que el robot se desplace a la izquierda y después se agache (figura 3.17). Por último, para pintar en la tercera fila será necesario que el robot de un paso atrás, se desplace a la izquierda y se agache (figura 3.17).

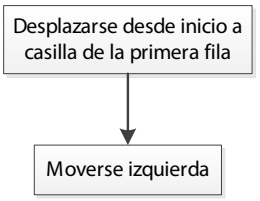



FIGURA 3.16: Jerarquía de primitivas: desplazamiento a casillas de primera fila.

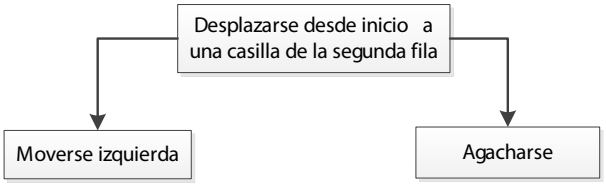



FIGURA 3.17: Jerarquía de primitivas: desplazamiento a casillas de segunda fila.

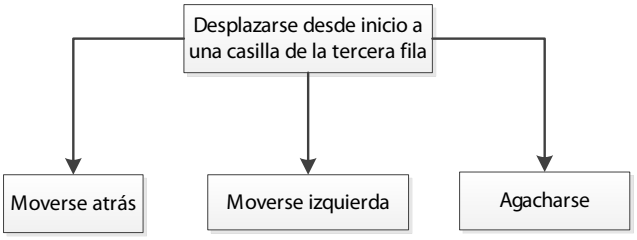



FIGURA 3.18: Jerarquía de primitivas: desplazamiento a casillas de tercera fila.

Los desplazamientos realizados a cualquier casilla implican que después el robot debe volver a una posición de origen. Para ello el robot deberá realizar los movimientos de desplazamiento a la inversa. Ésto queda reflejado en las figuras 3.19, 3.20 y 3.21.

La acción situada en el nodo raíz se ejecuta siguiendo el orden de los nodos hoja de izquierda a derecha. Para volver al origen después de dibujar en una casilla de la primera fila, el robot sólo tendrá que desplazarse a la derecha (figura 3.19). Por otro lado, si el robot ha pintado en la segunda fila será necesario que el robot se levante y desplace a la derecha (figura 3.20). Por último, para volver al origen cuando se pinta en la tercera fila será necesario que el robot se levante, de un paso adelante y se desplace a la derecha (figura 3.21).

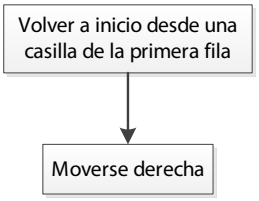



FIGURA 3.19: Jerarquía de primitivas: volver a origen desde casilla de primera fila.

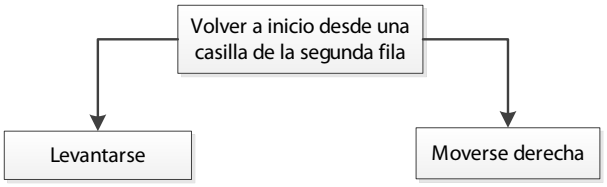



FIGURA 3.20: Jerarquía de primitivas: volver a origen desde casilla de segunda fila.


FIGURA 3.21: Jerarquía de primitivas: volver a origen desde casilla de tercera fila.

La figura 3.22 recoge las primitivas relacionadas con las habilidades de dibujo del robot. El dibujo de símbolos se ha descompuesto en trazos básicos para facilitar la generación de nuevos símbolos. Por otro lado, la primitiva situar rotulador permite al robot colocar el rotulador en la posición ideal para comenzar a pintar y la primitiva bajar rotulador devuelve el brazo a la posición inicial. El orden de ejecución de la acción *dibujar símbolo* es el de los nodos hijos de izquierda a derecha.

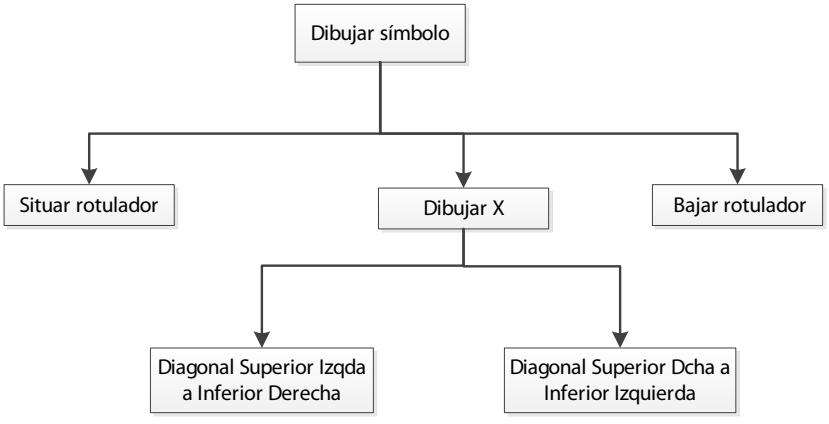



FIGURA 3.22: Estructura de acciones para dibujar.

Después de explicar los tres tipos de acciones disponibles, hay un nivel superior en el que se realiza la acción de dibujar un símbolo en una casilla. En este caso se realiza el proceso completo: el robot adquiere la pose INIT, se desplaza hasta la casilla dependiendo de la fila en la que se encuentre, dibuja el símbolo en la casilla que corresponda y vuelve a la posición inicial.

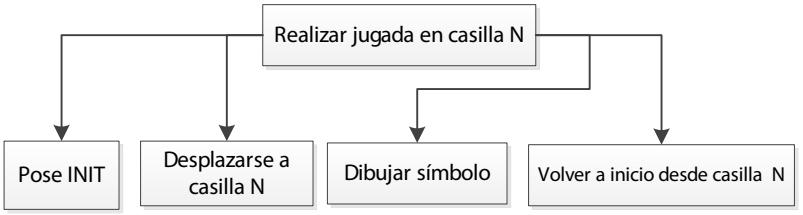



FIGURA 3.23: Estructura de acciones en nivel superior.

Para completar la explicación del módulo de dibujo hace falta definir la posición de inicio (origen) desde donde el robot comenzará a ejecutar siempre el primer movimiento. Se decidió utilizar la casilla 3 como referencia (primera fila, tercera columna). Ésto es porque, como el robot pinta con la mano derecha alejada del cuerpo, justamente en esa posición se encuentra el cuerpo centrado en el tablero y la mano derecha alineada con la tercera columna. Si el robot estira el brazo derecho llegará a la esquina superior derecha.

La figura 3.24 muestra lo que se ha explicado en el párrafo anterior. El recuadro A muestra el área de visión que tiene el robot con la cámara superior y el B muestra el de la cámara inferior. La letra X es el punto máximo al que llega el robot con el brazo derecho estirado. Hay otra anotación más sobre la posición de origen que no se aprecia en la imagen y es que las punteras del robot se deben encontrar a 8 cm. de la pared. La situación de los pies del robot junto a la medida de las punteras se controla con el folio DIN A-4 (figura 3.25), el cual contiene una plantilla con dichas referencias, y activando la Pose Init del robot para hacer coincidir los pies del robot con las huellas marcadas en el folio. Todas estas indicaciones sirven de referencia para colocar al robot en la posición esperada por el sistema.

#### 3.2.4.2. Módulo de visión

El módulo de visión tiene como misión principal identificar cuál ha sido la jugada del oponente. Para ello se desarrollan dos funcionalidades: la captura de imágenes y el análisis de cada una de ellas.

El robot capturará una imagen del estado de la casilla siempre que ésta estuviera vacía en la iteración anterior. La imagen se almacena con un nombre unívoco en el disco duro del ordenador donde se está ejecutando el módulo. Esto se repite hasta que todas las casillas vacías hayan sido fotografiadas. Todo esto se resume en el diagrama de actividad de la figura 3.26.

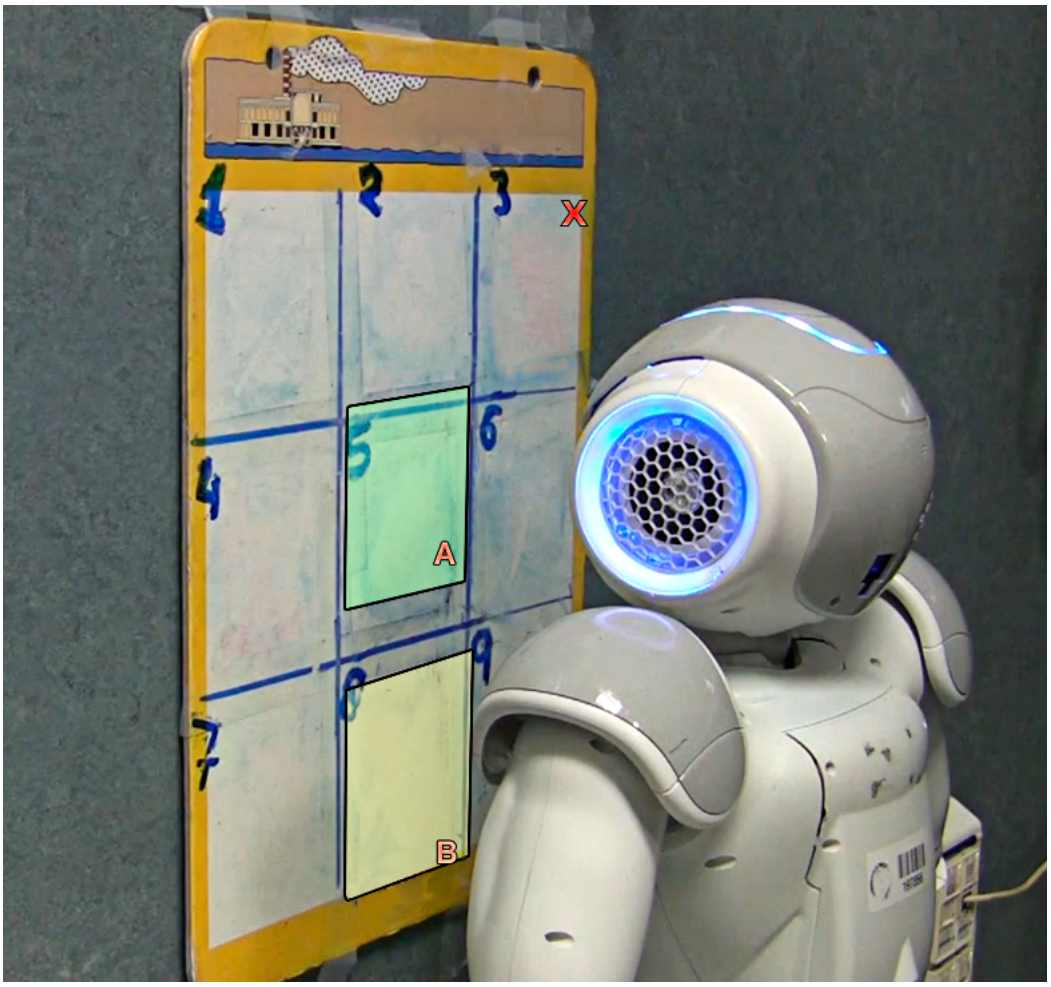



FIGURA 3.24: Análisis de la posición inicial (origen) del robot.

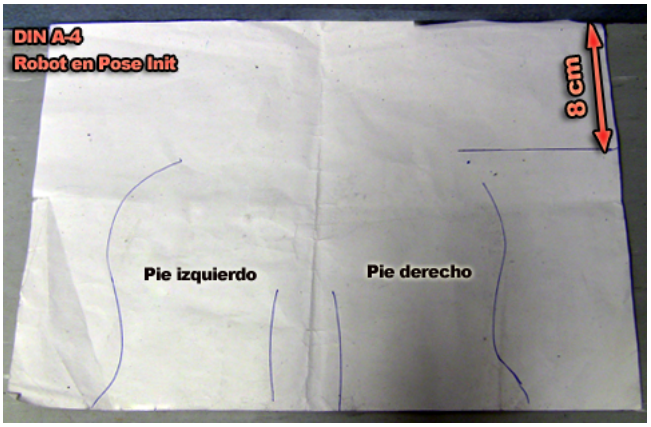



FIGURA 3.25: Plantilla para situar los pies del robot.

Para capturar una imagen, el robot NAO debe realizar previamente una secuencia de acciones que incluyen mover la cabeza (arriba, abajo, izquierda o derecha), agacharse, levantarse y acceder a la cámara superior e inferior. En los diagramas 3.27, 3.28 y 3.29 se muestra qué conjunto de acciones se utilizan según la fila en la que se encuentre la casilla a escanear. Dichas acciones se realizan siguiendo el orden de los respectivos nodos hijos, de izquierda a derecha.

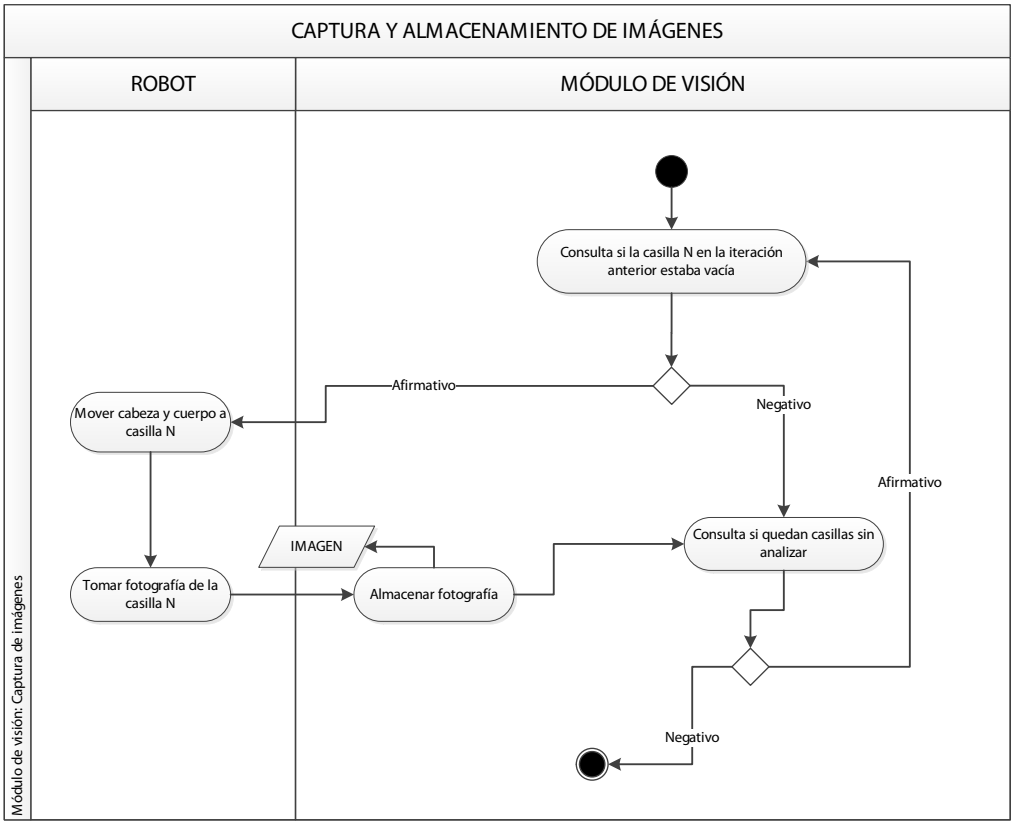



FIGURA 3.26: Diagrama de actividad: Capturar imagen.

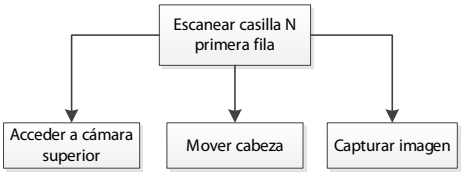



FIGURA 3.27: Jerarquía de primitivas: escanear casillas de la primera fila.

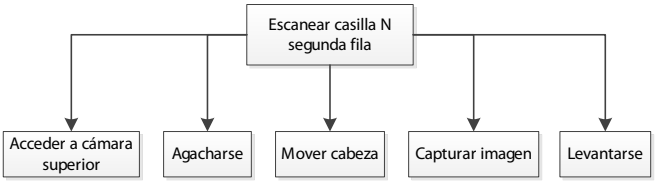



FIGURA 3.28: Jerarquía de primitivas: escanear casillas de la segunda fila.

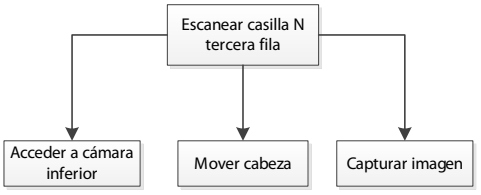



FIGURA 3.29: Jerarquía de primitivas: escanear casillas de la tercera fila.

Una vez se han capturado todas las imágenes, el sistema se encarga de obtener los histogramas de cada una de ellas. Mediante un bucle que comprueba qué histogramas no se han calculado aún, se llama a la función *histogram*. Ésta recibe una imagen como parámetro, la cual se lee y se convierte a escala de grises mediante la función de OpenCV *imread* [4].

Después se utiliza la función de OpenCV *calcHist* [5]que devuelve el histograma de la imagen en forma de lista de valores en la que cada posición equivale al número total de pixels de una luminosidad diferente presente en la imagen. En el trabajo se ha mantenido la longitud estándar de un histograma de estas características, 256 barras.

Finalmente, se compara cada par de histogramas de los estados de la casilla en el turno actual y en el anterior mediante la función *compareHist* [6], también de la librería OpenCV. Esta función ofrece cuatro métodos estadísticos diferentes para calcular la similitud de los histogramas: Correlación, Chi-Cuadrado, Intersección y distancia Bhattacharyya. De estos cuatro métodos, el de correlación y Bhattacharyya ofrecen valores de similitud en un rango $[0, 1]$, mientras que Chi-Cuadrado y la intersección lo hacen en un intervalo $[0, \infty]$.

Para decidir el método que se utilizaría para calcular la similitud de los histogramas, se probaron los cuatro métodos utilizando dos imágenes que eran iguales y otra tercera distinta de tal forma que se calculó la similitud de dos imágenes iguales y de dos diferentes. Cuando se analizaron los resultados de la similitud para cada uno de los cuatro métodos, se observó que, debido al rango $[0, 1]$ en el que trabajan los métodos de correlación y Bhattacharyya, los valores de similitud entre las imágenes iguales y las diferentes distaban muy poco en comparación con los valores que se obtenían con Chi-Cuadrado y la intersección. Pensando en el escenario que debía resolver esta parte del módulo de visión, que era ayudar a identificar la casilla que había pintado el contrincante, se obtendrían nueve valores de similitud de los cuales uno debe destacar muy por encima del resto. Esto hizo que se rechazara utilizar tanto correlación como Bhattacharyya, pues ya que la cámara del robot no tiene demasiados pixels y que las imágenes se convierten a escala de grises, los valores de similitud podrían confundirse demasiado en un intervalo $[0, 1]$ debido a su proximidad. Se reconocería mucho más rápido el par de imágenes diferente en un intervalo $[0, \infty]$.

Para tomar la decisión de utilizar intersección o Chi-Cuadrado, se analizó el significado que daba cada método a los extremos del rango $[0, \infty]$. En el caso de la intersección, cuanto más alto sea el valor obtenido, mayor es la similitud de los histogramas. Por otro lado, en Chi-Cuadrado, los valores crecientes indican menor similitud y un cero indica la máxima similitud. Esto último es lo más lógico de acorde a las características del sistema, pues de todos los pares de imágenes a analizar, tan sólo uno iba a mostrar signos de mínima similitud. Es más fácil identificar este caso

[4] http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html#imread

[5] http://docs.opencv.org/modules/imgproc/doc/histograms.html#calchist

[6] http://docs.opencv.org/modules/imgproc/doc/histograms.html#comparehist

particular si el objetivo es encontrar un único valor relativamente grande cuando la similitud desciende mientras que el resto de valores son cercanos a cero. Además, aunque se hable del valor de similitud, el objetivo es averiguar qué casilla es diferente o ha cambiado, y la secuencia lógica que pensaría la mayoría de usuarios sería que a mayor diferencia entre las imágenes habría que obtener un valor mayor que si ambas fuesen prácticamente iguales. Por todos estos motivos se escogió el método de Chi-Cuadrado.

Para calcular la similitud entre dos histogramas mediante Chi-Cuadrado hay utilizar la siguiente fórmula:

$$d(H_1, H_2) = \sum_{I=0}^{256} \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

- $H_n$: Histograma n
- $H_n(I)$: Valor de la barra I del histograma n

De todos los valores de similitud obtenidos, la casilla que ha pintado el jugador equivale a la que la que mayor distancia tenga con la imagen anterior.

Como es muy probable que en determinados casos el módulo de visión falle, se ha incluido una validación extra en la que en la misma terminal donde se ha lanzado el módulo, el sistema pedirá una confirmación sobre si la casilla que ha seleccionado él como jugada del contrincante es la correcta o no. Con introducir por teclado “si” o “no” es suficiente. Si la respuesta es “no”, el sistema pedirá que se introduzca el número de la casilla escogida por el adversario. Esto se explica con mayor detalle en B.

#### 3.2.4.3. Módulo de algoritmo del juego

Este módulo tiene por objetivo hallar la mejor jugada para el robot, entendiendo por mejor jugada aquella que le proporciona una ventaja mayor sobre su oponente. Como soporte al desarrollo se ha utilizado la librería AIMA-Python [26].

El presente módulo recibe como entrada la jugada que ha realizado el oponente en su turno así que lo primero que hace es actualizar la matriz equivalente al tablero del juego que se encuentra almacenada en el sistema. Como la nomenclatura utilizada por el humano y el robot son los números del 1 al 9, se traduce el valor de la casilla (entrada recibida) a la nomenclatura utilizada por el sistema (*fila, columna*).

| Nomenclatura Jugadores | Nomenclatura Sistema |
|---|---|
| 1 | (1, 1) |
| 2 | (1, 2) |
| 3 | (1, 3) |
| 4 | (2, 1) |
| 5 | (2, 2) |
| 6 | (2, 3) |
| 7 | (3, 1) |
| 8 | (3, 2) |
| 9 | (3, 3) |

TABLA 3.25: Equivalencia de nomenclaturas de las casillas

Ahora que ya se trabaja con la misma nomenclatura, se actualiza el tablero mediante la función de AIMA *result()*, que recibe por parámetro el tablero sin actualizar y la casilla con la que se quiere actualizar.

Para ayudar al espectador y al contrincante a entender el proceso que sigue el juego, se muestra por pantalla el tablero con la función *display()* de la librería AIMA. A continuación se puede ver un ejemplo del momento del juego descrito hasta este punto: el sistema ha identificado que la casilla que había escogido el oponente era la 6, actualiza el tablero y lo muestra.

```
---------------------
   It's number...6
---------------------
. . .
. . X
. . .
```

El siguiente paso es en el que interviene el algoritmo Minimax (sección 2.4) y la poda Alfa-Beta (sección 2.5). El sistema llama a la función *alphabeta_search()* pasándole por parámetro el tablero actualizado del juego e indicando que se trata de un tablero perteneciente al juego Tres en Raya. Esto último es porque AIMA también puede resolver otros juegos de tipo tablero pero las reglas que se aplican son diferentes, así que es importante especificarle que queremos que el tablero se evalúe con las heurísticas del Tres en Raya.

Dicha función llamará primero a la implementación del algoritmo minimax que desplegará un árbol de búsqueda como el que se muestra de ejemplo en la figura 3.30. Dada la situación en la que se encuentra el juego, que será el nodo raíz del árbol, el algoritmo irá expandiendo nodos hijos con las posibles continuaciones del juego. Cuando se alcance un nivel en el que no se pueden expandir más nodos porque el tablero está lleno, se comienzan a analizar esos nodos terminales evaluando qué jugador ganaría la partida. Se asigna un valor de los tres siguientes:

- -1 en caso de que ganase el oponente (el humano)
- 0 si empate
- 1 si gana el robot

Estos valores se propagan hacia el nodo raíz escogiendo el nodo con el mayor o menor valor, dependiendo de si el nivel pertenece a MAX o MIN. Cuando se haya alcanzado dicho nodo, el algoritmo devolverá la jugada que debe realizar el robot. Ésta será nuevamente traducida, esta vez a la nomenclatura del robot. Por último, el sistema actualizará el tablero del juego con la casilla elegida antes de dar paso a que NAO lo refleje sobre el tablero del entorno real.

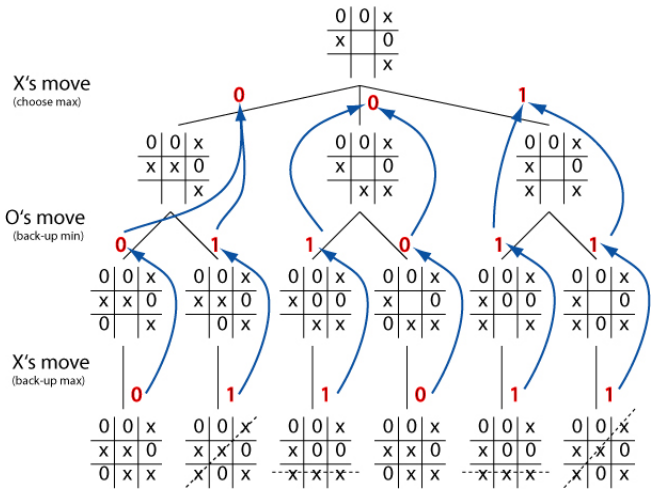



FIGURA 3.30: Árbol Minimax de Tres en Raya. (Fuente)

El siguiente fragmento equivale a lo que se imprime en la terminal cuando el algoritmo ha concluido la búsqueda del siguiente movimiento para el robot. En este ejemplo, NAO pintará el símbolo que haya escogido en la casilla 3 del tablero. Dicho símbolo no tiene por qué coincidir con los que se muestran por la terminal, es simplemente orientativo.

A la vez que se muestra este fragmento por la terminal, el robot comunica por voz que comienza su turno. Esto lo hace mediante la primitiva *hablar*.

```
--------------------
NAO it's your turn!
--------------------
(1, 3)
. . O
. . X
. . .
```

Una vez el robot ha terminado de pintar, indicará por voz al humano que comienza su turno diciendo "Human player it's your turn!".

# Capítulo 4

# Evaluación

En este capítulo se recogen todas las pruebas y experimentos realizados a la arquitectura presentada en el capítulo 3 y los resultados de una evaluación realizada a ocho personas ajenas al proyecto.

En las dos primeras secciones se plantea el entorno en el que se realizaron las pruebas, el tipo de pruebas que se realizaron y por último la descripción detallada de cada una de ellas y su relación con el catálogo de requisitos.

En la última sección del capítulo se describen los distintos escenarios y el cuestionario al que se sometieron los sujetos que evaluaron el sistema Tres en Raya.

## 4.1. Entorno de pruebas

Todas las pruebas y experimentos fueron realizados en el laboratorio 2.1.B16 de la Escuela Politécnica Superior (Universidad Carlos III de Madrid). Hay que destacar que el laboratorio no dispone de luz natural, así que esto era una gran ventaja para poder realizar los experimentos en cualquier momento sin que afectasen a la visión del robot, pues la cantidad de luz siempre era la misma.

En una de las paredes de dicho laboratorio se situó el tablero del juego, paralelo al suelo a 14cm. de éste. Alineado con el borde derecho del tablero, se sitúa en el suelo la referencia para colocar al robot en la posición inicial.

Después se colocó en la mano derecha del robot un rotulador de punta gorda apto para pizarra. Se fijó con celo para que no se escurriera, pues las manos del robot no pueden sujetarlo del todo bien.

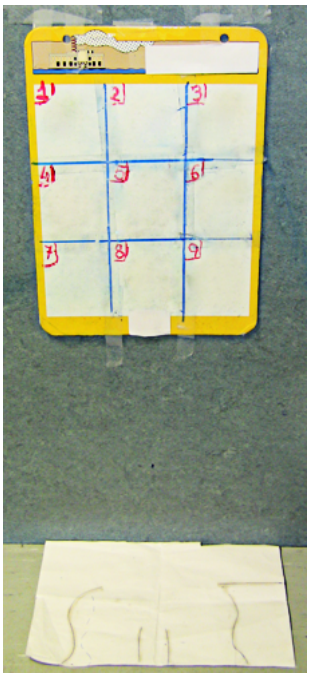


FIGURA 4.1: Entorno de pruebas I

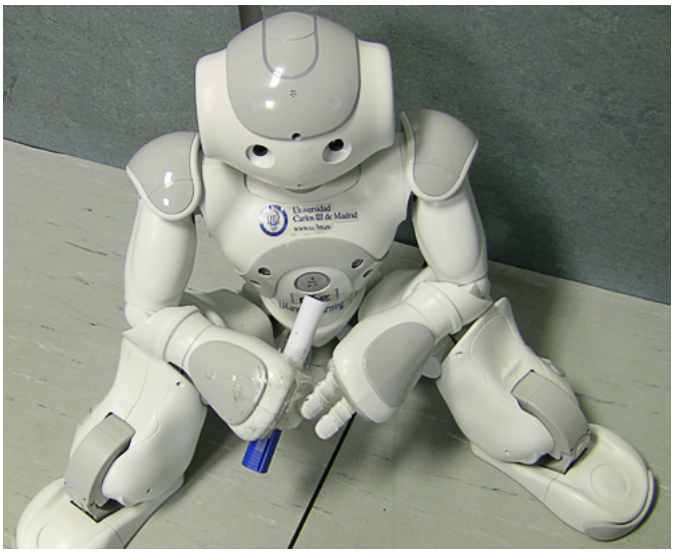


FIGURA 4.2: Entorno de pruebas II

Una vez se disponía de un ordenador con todas las dependencias instaladas, el código del sistema y el *router* encendido podría comenzarse cada uno de los test.

Este entorno en su totalidad sirvió para realizar todas las pruebas que incluían acciones del robot, como dibujar y desplazarse, y las de escaneo del tablero pertenecientes a la fase de visión. Para realizar las pruebas básicas del algoritmo del juego y del análisis de la jugada del contrincante únicamente se utilizó el ordenador y dos fotos de cada casilla tomadas por el robot.

## 4.2. Descripción de las pruebas

Para verificar el correcto funcionamiento del sistema, así como las respuestas esperadas a los objetivos marcados por el catálogo de requisitos, se establecen tres tipos de prueba: unitarias, de integración y de sistema.

- Unitarias: son aquellas que sirven para verificar el funcionamiento de métodos aislados o de pequeñas porciones de código. Son las primeras que se llevan a cabo de toda la especificación de pruebas.
- Integración: son aquellas que sirven para verificar el funcionamiento de pares o conjuntos de clases, módulos, partes etc. El caso que se presenta en este trabajo es el de la integración incremental: se comienza a probar primero dos clases, luego se le añade una tercera y así hasta que se realiza una prueba completa a todo el código.
- Sistema: son aquellas que sirven para verificar que los requisitos se cumplen en el código desarrollado. Cada una de ellas está relacionada con, al menos, un requisito del catálogo. Será imprescindible que toda esta batería de pruebas de resultados positivos porque son las que se suelen hacer delante del cliente.

En los siguientes apartados se describen en forma de tablas cada una de las pruebas a las que se ha sometido la arquitectura. Cada tabla está compuesta de la siguiente información:

**ID:** Identificador unívoco de la prueba. Todas seguirán la nomenclatura P(U/I/S)-XX donde U,I o S equivalen al tipo de prueba y XX equivale a un número de dos cifras comenzando por 01.

**Título:** Breve descripción de la prueba.

**Descripción:** Explicación detallada de la prueba

**Resultado:** Se explica en primer lugar si la prueba fue exitosa y si hace falta puntualizar algún incidente o información extra.

### 4.2.1. Pruebas unitarias

| | |
|---|---|
| **ID** | PU-01 (VIDEO) |
| **Título** | Dibujar símbolo X en primera fila, segunda y tercera |
| **Descripción** | Verificar que el robot puede dibujar el símbolo X en cualquier fila situándolo previamente en la posición que corresponda. |
| **Métodos** | `draw_Half_X_UpperRight_firstRow()`, `draw_Half_X_UpperLeft_firstRow()`, `draw_Half_X_UpperRight_thirdRow()`, `draw_Half_X_UpperLeft_thirdRow()`, `draw_Half_X_UpperRight_secondRow()`, `draw_Half_X_UpperLeft_secondRow()` |
| **Resultado** | Éxito |

TABLA 4.1: Prueba PU-01: Dibujar símbolo X en fila

| | |
|---|---|
| **ID** | PU-02 |
| **Título** | El robot se agacha |
| **Descripción** | Se comprueba que el robot se agacha de forma autónoma sin caerse. |
| **Métodos** | `down()` |
| **Resultado** | Éxito |

TABLA 4.2: Prueba PU-02: El robot se agacha

| | |
|---|---|
| **ID** | PU-03 |
| **Título** | El robot vuelve a posición inicial estando agachado |
| **Descripción** | Se comprueba que el robot vuelve a su posición inicial de forma autónoma estando inicialmente agachado sin caerse. |
| **Métodos** | `PoseInit()` |
| **Resultado** | Éxito |

TABLA 4.3: Prueba PU-03: El robot vuelve a pose inicial

| | |
|---|---|
| **ID** | PU-04 |
| **Título** | El robot coloca el rotulador en posición para dibujar |
| **Descripción** | Se comprueba que el robot es capaz de colocar el rotulador para poder dibujar un símbolo en cualquier fila. Tanto si el robot está agachado como si está de pie. |
| **Métodos** | `hand_init(), hand_init_from_down()` |
| **Resultado** | Éxito |

TABLA 4.4: Prueba PU-04: Rotulador en posición para dibujar

| | |
|---|---|
| **ID** | PU-05 |
| **Título** | El robot coloca el rotulador en posición inicial |
| **Descripción** | Se comprueba que el robot es capaz de colocar el rotulador en su posición inicial desde la posición para dibujar. Tanto si el robot está agachado como si está de pie. |
| **Métodos** | `hand_down(), hand_down_from_down()` |
| **Resultado** | Éxito |

TABLA 4.5: Prueba PU-05: Rotulador en posición inicial

| | |
|---|---|
| **ID** | PU-06 |
| **Título** | Desplazamientos del robot |
| **Descripción** | Se comprueba que el robot es capaz de moverse hacia la izquierda, derecha, adelante y atrás. |
| **Métodos** | `walk()` |
| **Resultado** | Éxito parcial. El robot se desvía ligeramente en sus desplazamientos. |

TABLA 4.6: Prueba PU-06: Desplazamientos del robot

| | |
|---|---|
| **ID** | PU-07 |
| **Título** | El robot habla |
| **Descripción** | Se comprueba que el robot es capaz de transmitir las frases que le llegan. |
| **Métodos** | `talk()` |
| **Resultado** | Éxito |

TABLA 4.7: Prueba PU-07: El robot habla

| | |
|---|---|
| **ID** | PU-08 |
| **Título** | El robot intercambia la cámara activa |
| **Descripción** | El robot es capaz de cambiar la cámara superior por la inferior y viceversa. |
| **Métodos** | `changeCamera()` |
| **Resultado** | Éxito |

TABLA 4.8: Prueba PU-08: El robot intercambia la cámara activa

| | |
|---|---|
| **ID** | PU-09 |
| **Título** | El robot captura una foto |
| **Descripción** | Se comprueba que el robot es capaz de capturar dos fotos del entorno, una con cada cámara. |
| **Métodos** | `takeImage()` |
| **Resultado** | Éxito |

TABLA 4.9: Prueba PU-09: El robot captura dos fotos

| | |
|---|---|
| **ID** | PU-10 |
| **Título** | El sistema genera el histograma de una foto |
| **Descripción** | Se recibe una foto como entrada y el sistema genera el histograma de la misma. |
| **Métodos** | `histogram()` |
| **Resultado** | Éxito |

TABLA 4.10: Prueba PU-10: El sistema genera el histograma de una foto

| | |
|---|---|
| **ID** | PU-11 |
| **Título** | El sistema compara dos histogramas |
| **Descripción** | Se reciben dos histogramas, primero dos iguales y luego dos diferentes y el sistema calcula la similitud de sus histogramas. |
| **Métodos** | `compareHistogram()` |
| **Resultado** | Éxito |

TABLA 4.11: Prueba PU-11: El sistema compara dos histogramas

### 4.2.2. Pruebas de integración

| | |
|---|---|
| **ID** | PI-01 |
| **Título** | El robot dibuja el símbolo X en una casilla N |
| **Descripción** | Se le indica al robot una casilla en la que pintar el símbolo X y él mismo de forma autónoma es capaz de desplazarse, colocarse en la posición adecuada y pintarlo. |
| **Métodos** | `moveToCell(), drawX()` |
| **Resultado** | Éxito |

TABLA 4.12: Prueba PI-01: El robot dibuja el símbolo X en una casilla N

| | |
|---|---|
| **ID** | PI-02 |
| **Título** | El robot vuelve a su posición inicial tras dibujar el símbolo X en una casilla N |
| **Descripción** | El robot es capaz de bajar el rotulador y colocarse correctamente para desplazarse al inicio. |
| **Métodos** | `backToStart()` |
| **Resultado** | Éxito |

TABLA 4.13: Prueba PI-02: El robot vuelve a su posición inicial

| | |
|---|---|
| **ID** | PI-03 |
| **Título** | Módulo de dibujo completo |
| **Descripción** | El robot es capaz de realizar el conjunto de movimientos completo: se desplaza a la casilla, coloca el rotulador, pinta el símbolo, baja el rotulador y vuelve a la posición de la que partió. |
| **Métodos** | `print_symbol()` |
| **Resultado** | Éxito |

TABLA 4.14: Prueba PI-03: Módulo de dibujo completo

| | |
|---|---|
| **ID** | PI-04 |
| **Título** | Observar casilla N y tomar fotografía |
| **Descripción** | El robot es capaz de realizar los movimientos de cabeza y de cámaras para tomar una fotografía de la casilla que se le haya indicado. |
| **Métodos** | `scanCell()` |
| **Resultado** | Éxito |

TABLA 4.15: Prueba PI-04: Observar casilla N y tomar fotografía

| | |
|---|---|
| **ID** | PI-05 |
| **Título** | Escanear el tablero completo |
| **Descripción** | El robot es capaz de realizar los movimientos de cabeza, cuerpo y de cámaras para tomar una fotografía de cada casilla del tablero. |
| **Métodos** | `scanBoard()` |
| **Resultado** | Éxito |

TABLA 4.16: Prueba PI-05: Escanear el tablero completo

| | |
|---|---|
| **ID** | PI-06 |
| **Título** | Calcular similitud entre las dos primeras rondas |
| **Descripción** | El robot escanea el tablero una vez, después un humano pinta un símbolo en alguna casilla y el robot escanea de nuevo el tablero. De las 18 fotos que ha tomado, dos por casilla, extraerá, por medio de los histogramas, la que ha cambiado |
| **Métodos** | `scanBoard(), generate_and_Compare_Histograms()` |
| **Resultado** | Éxito |

TABLA 4.17: Prueba PI-06: Calcular similitud entre las dos primeras rondas

| | |
|---|---|
| **ID** | PI-07 |
| **Título** | Calcular similitud entre dos rondas en mitad de la partida |
| **Descripción** | Después de jugar dos rondas, el tablero está semi-completo. El robot escanea el tablero una vez. Se fija sólamente en las casillas vacías. Después un humano pinta un símbolo en alguna casilla libre y el robot escanea de nuevo el tablero tomando tantas fotos como casillas quedaran libres en el primer escaneo. De ese conjunto de pares de fotos extraerá, por medio de los histogramas, la que ha cambiado |
| **Métodos** | `scanBoard(), generate_and_Compare_Histograms()` |
| **Resultado** | Éxito |

TABLA 4.18: Prueba PI-07: Calcular similitud entre dos rondas

| | |
|---|---|
| **ID** | PI-08 |
| **Título** | Módulo de dibujo y visión integrados |
| **Descripción** | El robot dibuja en una casilla N, después escanea el tablero. Cuando ha terminado, un humano pinta un símbolo en otra casilla y el robot escanea de nuevo el tablero. El robot identificará la que ha elegido el humano. |
| **Métodos** | `print_symbol(), scanBoard(), generate_and_Compare_Histograms()` |
| **Resultado** | Éxito |

TABLA 4.19: Prueba PI-08: Módulo de dibujo y visión integrados

| | |
|---|---|
| **ID** | PI-09 |
| **Título** | Actualizar tablero y sugerir jugada al robot |
| **Descripción** | El sistema actualiza el tablero en función de la casilla que ha ocupado el humano (se simula) y, en función del estado del tablero, el algoritmo le indica al robot la casilla en la que debe dibujar. |
| **Métodos** | `result(), alphabeta_search()` |
| **Resultado** | Éxito |

TABLA 4.20: Prueba PI-09: Actualizar tablero y sugerir jugada al robot

| | |
|---|---|
| **ID** | PI-10 (VIDEO) |
| **Título** | Módulo de dibujo, visión y algoritmo integrados |
| **Descripción** | Se realiza una iteración completa del juego: el robot escanea el tablero, el humano pinta en una casilla, el robot vuelve a escanear e identifica la casilla que ha pintado el humano, se actualiza el tablero, el algoritmo le sugiere dice al robot qué casilla pintar y el robot se desplaza, la pinta y vuelve a la posición inicial. |
| **Métodos** | `print_symbol()`, `scanBoard()`, `generate_and_Compare_Histograms()`, `result()`, `alphabeta_search()` |
| **Resultado** | Éxito |

TABLA 4.21: Prueba PI-10: Módulo de dibujo, visión y algoritmo integrados

### 4.2.3. Pruebas de sistema

| | |
|---|---|
| **ID** | PS-01 |
| **Título** | Sistema Tres en Raya: comienza jugador humano |
| **Descripción** | Comienza el jugador humano a pintar y después lo hará el robot. Se irá desarrollando todo el juego como se indicó en 4.20 hasta que se llegue a un empate o gane uno de los jugadores. |
| **Resultado** | Éxito |

TABLA 4.22: Prueba PS-01: Sistema Tres en Raya: comienza jugador humano

| | |
|---|---|
| **ID** | PS-02 |
| **Título** | Sistema Tres en Raya: comienza jugador NAO |
| **Descripción** | Comienza NAO a pintar y después lo hará el jugador humano. Se irá desarrollando todo el juego como se indicó en 4.20 hasta que se llegue a un empate o gane uno de los jugadores. |
| **Resultado** | Éxito |

TABLA 4.23: Prueba PS-02: Sistema Tres en Raya: comienza jugador NAO

| | |
|---|---|
| **ID** | PS-03 |
| **Título** | Sistema Tres en Raya: el sistema echa a suertes quien empieza |
| **Descripción** | Se irá desarrollando todo el juego como se indicó en 4.20 hasta que se llegue a un empate o gane uno de los jugadores. |
| **Resultado** | Éxito |

TABLA 4.24: Prueba PS-03: Sistema Tres en Raya: el sistema echa a suertes quien empieza

## 4.3. Cuestionario de evaluación

Con el objetivo de probar el sistema desarrollado en un entorno real, se contactó con nueve personas de rango de edad 20-30 años para que jugasen una o dos partidas al Tres en Raya con NAO. Los escenarios que se propusieron fueron:

- **ESCENARIO 1:** Juega una partida al Tres en Raya en la que empieza el humano a escoger casilla.
- **ESCENARIO 2:** Juega una partida al Tres en Raya en la que empieza el robot a escoger casilla.
- **ESCENARIO 3:** Juega una partida al Tres en Raya y déjate ganar por el robot.

Los escenarios 1 y 2 son los que mejor simulan un entorno real pero se añadió el escenario 3 para comprobar el correcto funcionamiento del algoritmo en dicho entorno real.

Las nueve personas jugaron un total de 11 partidas, de las cuales se grabaron en vídeo seis. La tabla 4.25 muestra qué se realizó en cada prueba.

| | | |
|---|---|---|
| Sujeto 1 (Mujer) | ESCENARIO 1 | - |
| Sujeto 2 (Mujer) | ESCENARIO 3 | VIDEO |
| Sujeto 3 (Hombre) | ESCENARIO 3 | VIDEO |
| Sujeto 4 (Hombre) | ESCENARIO 1 | VIDEO 1 VIDEO 2 |
| Sujeto 5 (Hombre) | ESCENARIO 2 | VIDEO |
| Sujeto 6 (Hombre) | ESCENARIO 2 | - |
| Sujeto 7 (Hombre) | ESCENARIO 1 y 2 | VIDEO ESCENARIO 1 |
| Sujeto 8 (Hombre) | ESCENARIO 1 | - |
| Sujeto 9 (Hombre) | ESCENARIO 1 y 2 | - |

TABLA 4.25: Pruebas realizadas en la evaluación.

En ninguna de las partidas jugadas el humano consiguió vencer a NAO independientemente de quién comenzase a jugar en primer lugar. Los movimientos del robot son perfectos y no dejan escapar ninguna oportunidad de ganar o de empatar. Respecto al módulo de visión, se puede apreciar en la tabla 4.26 que se ha comportado bastante bien. De media se comete un fallo y, aunque no se refleja en la tabla, siempre es durante las dos primeras iteraciones del juego. Esto es porque como se va reduciendo el abanico de casillas a identificar, la tasa de acierto aumenta proporcionalmente a cada iteración de juego.

| | | |
|---|---|---|
| Sujeto 1 | ACIERTOS VISIÓN: 4/4 | - |
| Sujeto 2 | ACIERTOS VISIÓN: 3/4 | Casilla 9 |
| Sujeto 3 | ACIERTOS VISIÓN: 3/4 | Casilla 3 |
| Sujeto 4 | ACIERTOS VISIÓN: 3/4 y 3/4 | Casillas 7 y 4 |
| Sujeto 5 | ACIERTOS VISIÓN: 4/4 | - |
| Sujeto 6 | ACIERTOS VISIÓN: 3/4 | Casilla 9 |
| Sujeto 7 | ACIERTOS VISIÓN: 4/4 y 2/3 | Casilla 9 |
| Sujeto 8 | ACIERTOS VISIÓN: 3/4 | Casilla 7 |
| Sujeto 9 | ACIERTOS VISIÓN: 3/4 y 2/4 | Casillas 5, 3 y 8 |

TABLA 4.26: Análisis de fallos del módulo de visión.

Después de esta primera parte, se solicitó a las personas participantes que rellenasen un cuestionario para reflejar sus impresiones, problemas y sugerencias de mejora. El cuestionario equivale a las figuras 4.3 y 4.4. Éste contiene preguntas de respuesta libre y de respuesta en forma de escala Likert (1 al 5). Se han escogido cinco grados porque está demostrado que son los que reflejan mejor las sensaciones. El tercer punto en la escala siempre equivale a un neutro o *mid-point*.

Era importante analizar las consecuencias que tendría la presencia del *mid-point* en el cuestionario, ya que se tiende a elegir el neutro cuando no ha quedado clara una pregunta o cuando no se sabe qué contestar. Esto se ha compensado alternando adjetivos positivos y negativos, preguntas positivas y negativas y preguntas similares aunque con diferente matiz en distintos puntos del cuestionario. De esta forma se puede trabajar cómodamente con una escala de grado cinco sin que afecte demasiado el neutro a los resultados finales y se mantiene la atención del encuestado por los cambios en las preguntas de connotación positiva y negativa.

Finalmente, destacar que el cuestionario se ha dividido en tres partes: una relacionada con la partida en general, otra específica sobre el robot y la última para transmitir sugerencias de mejora a tener en cuenta en las líneas futuras del trabajo.

# Evaluación del módulo 'Tres en Raya'

Después de haber jugado al juego Tres en Raya con el robot NAO, agradecería que rellenases el siguiente cuestionario para transmitir tus impresiones y sugerencias de mejora.
*Obligatorio

## Sobre la partida

Evalúa la dificultad del juego en general *

1 2 3 4 5

Muy fácil ○ ○ ○ ○ ○ Muy difícil

El proceso de dibujar un símbolo en una casilla fue... *

1 2 3 4 5

Muy fácil ○ ○ ○ ○ ○ Muy complicado

La partida te pareció... *

| | Muy en desacuerdo | En desacuerdo | Neutro | De Acuerdo | Muy de acuerdo |
|---|---|---|---|---|---|
| Larga | ○ | ○ | ○ | ○ | ○ |
| Entretenida | ○ | ○ | ○ | ○ | ○ |
| Complicada de ganar | ○ | ○ | ○ | ○ | ○ |
| Aburrida | ○ | ○ | ○ | ○ | ○ |
| Fácil | ○ | ○ | ○ | ○ | ○ |
| Lenta | ○ | ○ | ○ | ○ | ○ |

## Sobre el robot

La habilidad de NAO para jugar al Tres en Raya me pareció... *

1 2 3 4 5

Muy mala ○ ○ ○ ○ ○ Muy buena

FIGURA 4.3: Cuestionario de evaluación I.

La habilidad de visión de NAO para identificar la casilla del contrincante me pareció... *

1 2 3 4 5

Muy mala ○ ○ ○ ○ ○ Muy buena

La habilidad de dibujar de NAO me pareció... *

1 2 3 4 5

Muy mala ○ ○ ○ ○ ○ Muy buena

## Sugerencias de mejora

¿Qué sería lo siguiente que mejorarías del módulo Tres en Raya? *

¿Qué es lo que más te gustó? *

¿Qué echaste de menos en el juego?

De las siguientes opciones, escoge la que más interesante te parezca para incluir en el módulo Tres en Raya *

FIGURA 4.4: Cuestionario de evaluación II.

## 4.4. Resultados obtenidos

En esta sección se analizan los resultados obtenidos en el cuestionario de evaluación del sistema Tres en Raya.

En primer lugar se muestran los resultados relacionados con la partida que había jugado cada persona (figuras 4.5, 4.6, 4.7 y 4.8). Después se muestran los relativos a las habilidades del robot (figura 4.9). Sobre los resultados de las dos primeras preguntas, el juego en sí no tiene una

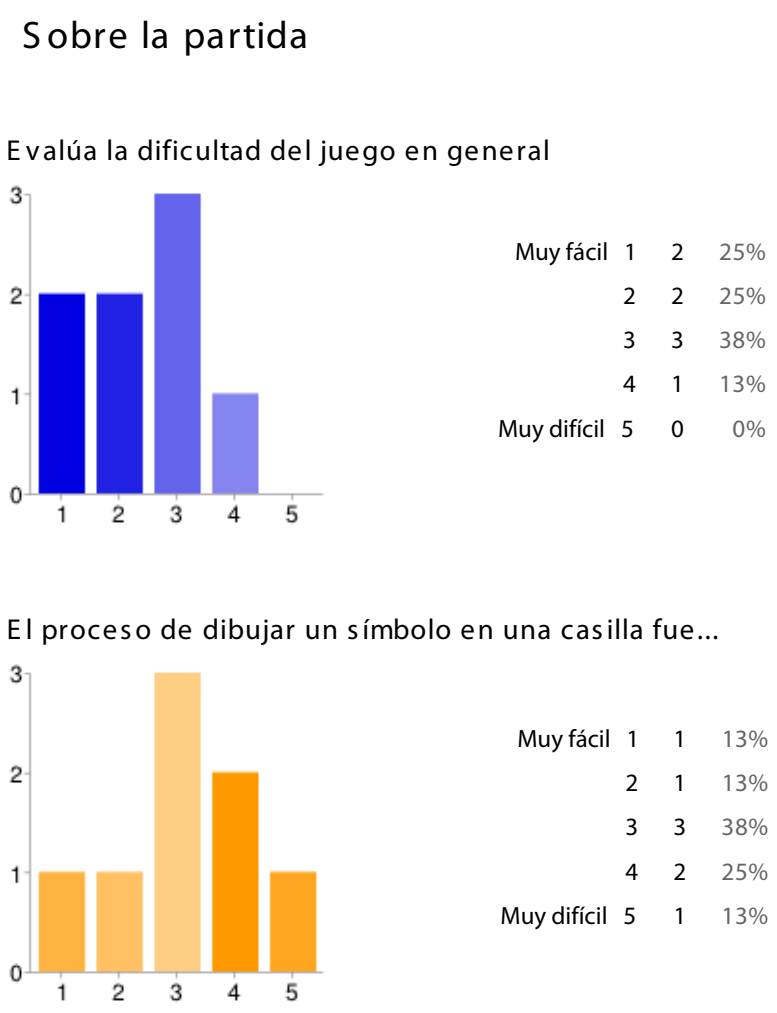



FIGURA 4.5: Resultados de la evaluación I.

dificultad excesiva, por lo que se esperaba que al menos la mitad de los encuestados escogieran los valores del 1 al 3. Respecto a cómo pintar un símbolo en una casilla, los resultados tienden hacia los medios altos ya que sí que es cierto que para pintar las casillas de las filas inferiores hace falta algo de maña porque el robot se queda en medio y muy pegado al tablero.

De las siguientes preguntas (figura 4.6) se percibe cómo la partida se hace larga, sobre todo por la lentitud del reconocimiento de la jugada del humano pero aún así los encuestados reconocen

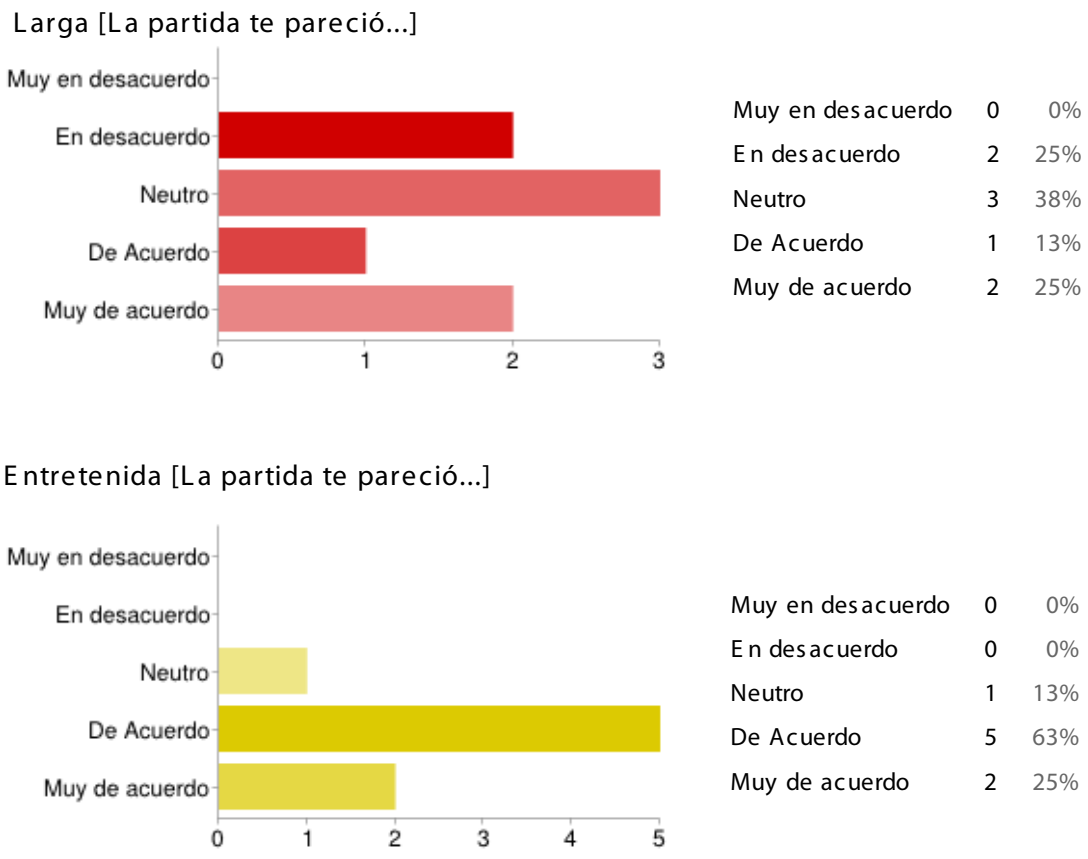
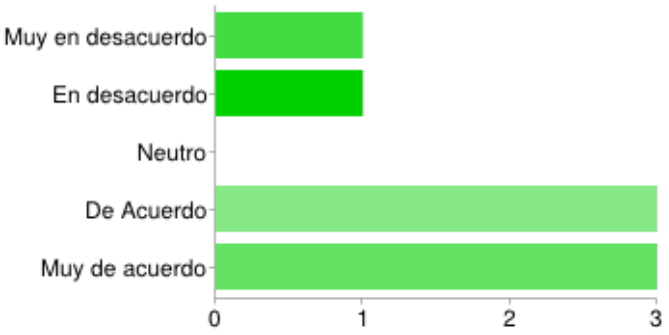



FIGURA 4.6: Resultados de la evaluación II.

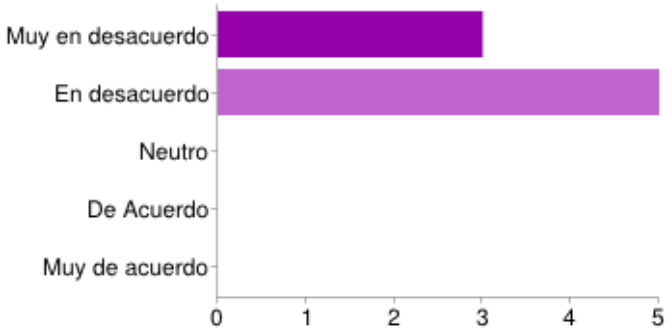
que la partida es entretenida. Durante las diferentes evaluaciones, los encuestados se han mostrado muy contentos y curiosos al enfrentarse ante un robot, la mayoría lo califica como una experiencia muy buena y diferente. Además al preguntar lo contrario, si la partida resultó aburrida (pues se sabía que la lentitud del robot iba a ser lo más perjudicial) todos los encuestados lo negaron. En la mayoría de los casos, como refleja la figura 4.7, la partida resultó complicada de ganar.

Complicada de ganar [La partida te pareció...]




| | | |
|---|---|---|
| Muy en desacuerdo | 1 | 13% |
| En desacuerdo | 1 | 13% |
| Neutro | 0 | 0% |
| De Acuerdo | 3 | 38% |
| Muy de acuerdo | 3 | 38% |

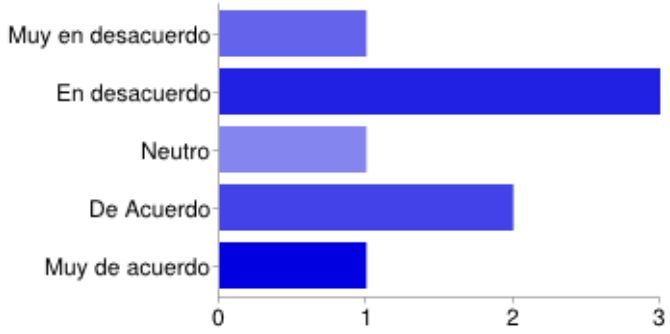
Aburrida [La partida te pareció...]




| | | |
|---|---|---|
| Muy en desacuerdo | 3 | 38% |
| En desacuerdo | 5 | 63% |
| Neutro | 0 | 0% |
| De Acuerdo | 0 | 0% |
| Muy de acuerdo | 0 | 0% |

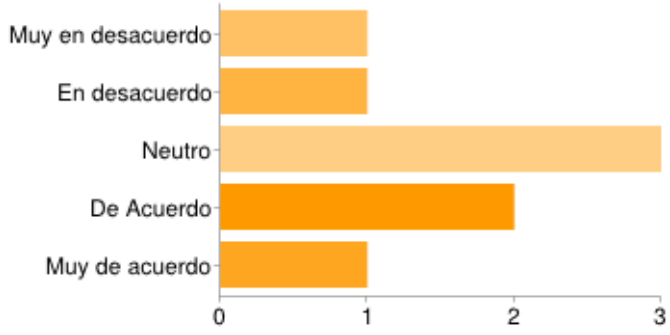
FIGURA 4.7: Resultados de la evaluación III.

Fácil [La partida te pareció...]




| | | |
|---|---|---|
| Muy en desacuerdo | 1 | 13% |
| En desacuerdo | 3 | 38% |
| Neutro | 1 | 13% |
| De Acuerdo | 2 | 25% |
| Muy de acuerdo | 1 | 13% |

Lenta [La partida te pareció...]




| | | |
|---|---|---|
| Muy en desacuerdo | 1 | 13% |
| En desacuerdo | 1 | 13% |
| Neutro | 3 | 38% |
| De Acuerdo | 2 | 25% |
| Muy de acuerdo | 1 | 13% |

FIGURA 4.8: Resultados de la evaluación IV.

Respecto a los adjetivos “lenta” y “fácil”, el segundo es el más difícil de analizar por lo que los encuestados entienden por fácil. Muchos lo han relacionado con el concepto de ser capaz de ganar la partida y otros con la dificultad que tiene jugar el tres en raya contra un robot, por eso las respuestas son dispares. Con el adjetivo “Lenta” hay diversidad de opiniones pero la mayoría se sitúan en la escala medio-alto.

Las habilidades del robot son lo que mejor han valorado los encuestados y también lo que más llama la atención (figura 4.9). La habilidad para escoger el siguiente movimiento es la que se sitúa en primer lugar quedando después empatadas la de visión y la de dibujar.

Respecto a las sugerencias de mejora, se ha elaborado un resumen de las respuestas obtenidas para cada pregunta.

**¿Qué sería lo siguiente que mejorarías?**

- La capacidad de desplazarse de NAO entre las diferentes casillas. Fue lo más reclamado por los encuestados.
- Optimizar el proceso de visión para que la partida sea más rápida
- Retirarse hacia atrás para que el humano pinte con mayor comodidad
- Que el humano pueda confirmar la casilla identificada por el robot por voz en vez de por teclado

**¿Qué es lo que más te gustó?**

- Que el Nao dijera cuando era los turnos y cuando ya ha terminado el juego.
- La visión artificial de NAO para ser capaz de identificar donde pintar y pintar el solo.
- El algoritmo que sigue para detectar las casillas pintadas.
- La capacidad para localizar las casillas y escribir en ellas.
- Que el robot sea capaz de dibujar las equis!!

**¿Qué echaste de menos en el juego?**

- Me pareció que estaba todo muy bien conseguido. Nada en especial.
- La colocación automática del robot en todo momento, aunque reconozco que es una tarea muy difícil.
- Moverse hacia un lado para facilitar la jugada del usuario.

- Agilidad
- Algo de aleatoriedad a la hora de posicionar la casilla cuando empieza el robot a jugar o en el caso de que las opciones ganadoras sean iguales.

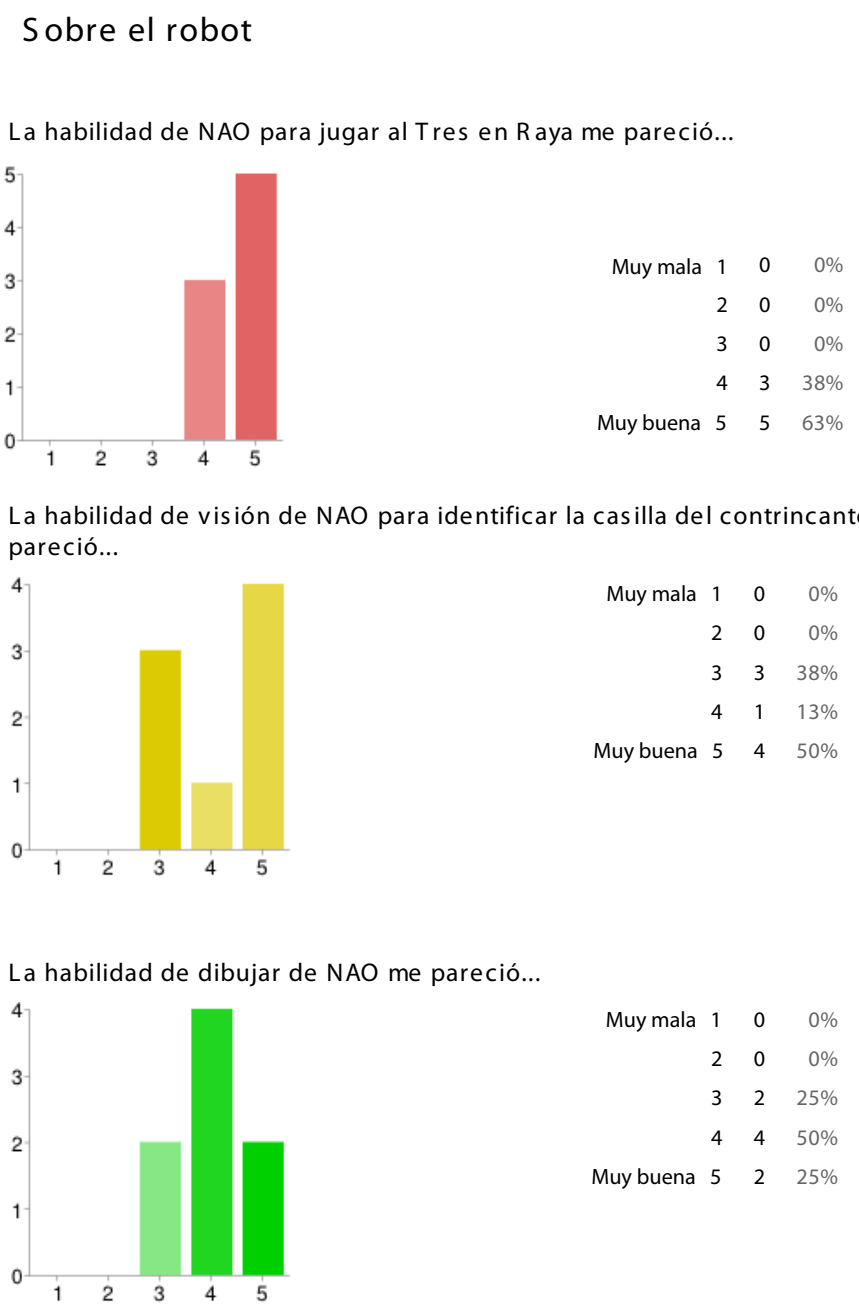
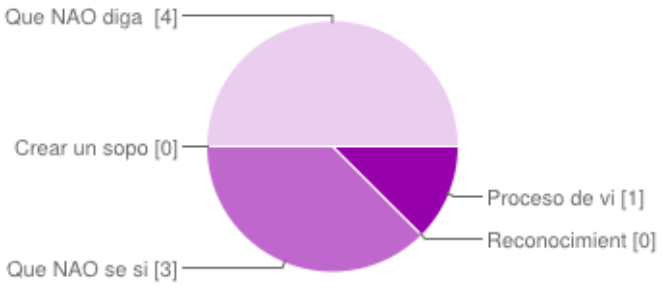



FIGURA 4.9: Resultados de la evaluación V.

Por último, para dar por terminada la parte de sugerencias se le proponía al encuestado cinco alternativas diferentes de mejora del sistema Tres en Raya de la cual debía escoger la que más le gustase (figura 4.10). Algunas alternativas ya las habían sugerido los propios encuestados en las

preguntas de respuesta libre, pero llama la atención que la mitad de los encuestados escogiera la propuesta de *"Que NAO diga frases graciosas en función del estado del juego"*, pues se ve claramente cómo los usuarios prefieren la interacción y socialización del robot antes que la perfección de cada movimiento.




| | | |
|---|---|---|
| Proceso de visión por reconocimento de símbolos | 1 | 13% |
| Reconocimiento por voz para verificar la identificación de la casilla | 0 | 0% |
| Que NAO se sitúe automáticamente en cada posición | 3 | 38% |
| Crear un soporte para el rotulador | 0 | 0% |
| Que NAO diga frases graciosas en función del estado del juego | 4 | 50% |

FIGURA 4.10: Resultados de la evaluación VI.

# Capítulo 5

# Gestión del trabajo

En este capítulo queda reflejada la planificación realizada para llevar a cabo el desarrollo íntegro del proyecto. Después se muestra el cálculo del presupuesto total y desglosado en base a los costes generados por el mismo.

## 5.1. Planificación

Esta sección primero se recoge la justificación de la metodología utilizada para desarrollar el trabajo y después se explica en qué consiste. Posteriormente, se detalla la estimación de tiempos y se presenta un diagrama de Gantt que muestra toda la información en conjunto.

### 5.1.1. Justificación de la metodología

Debido a las características que presenta este trabajo, dividido claramente en tres fases: dibujo, visión y búsqueda, era necesario disponer de una metodología iterativa que permitiera desarrollar cada una de las fases de forma independiente para, en la etapa final, unirlas y obtener el producto final. Se estudiaron tres metodologías: Prototipado, Incremental y Espiral. La primera de ellas no se adaptaba nada bien al proyecto porque exigía la construcción de un prototipado en poco tiempo. Con la segunda sucedía algo similar, compuesta de dos etapas: inicialización e iterativa, ya en la primera etapa recomendaba la elaboración de un prototipo con el que poder interactuar para después mejorarlo. Finalmente, el modelo en espiral sí que permitía fragmentar el proyecto en fases casi independientes, realizando para cada una de ellas análisis, diseño, desarrollo, pruebas y documentación. Aunque este modelo es más lento en cuanto a desarrollo que los otros dos, por la ventaja que se han comentado se decidió aplicarlo a todo el proceso.

### 5.1.2. El modelo en espiral

El modelo en espiral de desarrollo de software fue ideado por Barry W. Boehm en 1986 [27]. El principal objetivo del modelo es dividir un proyecto en partes más pequeñas. Cada una de estas partes deberá pasar por todas las fases:

- Fijar objetivos, plantear alternativas e identificar restricciones
- Evaluar alternativas, identificar y resolver riesgos
- Desarrollar, hacer test y pruebas
- Planificar la siguiente etapa de trabajo

Estos cuatro puntos equivalen a una iteración completa, es decir, a una vuelta de la espiral. Según en qué sitio de la espiral se encuentre el trabajo, hay que realizar unas tareas u otras. Cada tarea pertenece a uno de los siguientes grupos: análisis, diseño, codificación, pruebas y documentación. Si bien es cierto, dependiendo de qué etapa del proyecto se analice, unas tareas tienen mayor importancia que otras, por ejemplo al inicio del proyecto se hizo mayor hincapié en las tareas de análisis que en las de pruebas porque el trabajo requería de una investigación previa.

Las tareas de análisis consisten principalmente en identificación de requisitos, estudio de tecnologías y estado del arte, análisis de riesgos, estudio de alternativas y restricciones, establecer el alcance de cada módulo etc.

Las tareas de diseño consisten en la elaboración de todo tipo de diagramas que permitan acercar el trabajo realizado en el análisis a la fase de codificación. También en este punto se toman decisiones respecto al futuro funcionamiento del código.

Las tareas de codificación incluyen el desarrollo del código del proyecto y el trabajo realizado con los simuladores y el *hardware* del robot.

Las tareas de pruebas consisten en la elaboración de una batería de pruebas que permita verificar el buen funcionamiento del módulo o el sistema completo. Habrá una batería de pruebas diferente según los tipos de prueba establecidos en el capítulo 4.

Las tareas de documentación son la redacción de las diferentes partes de la memoria del trabajo y cualquier tipo de elaboración de manuales o Wikis que sirvan para recoger todo el conocimiento aprendido durante el desarrollo del proyecto.

### 5.1.3. Distribución de tareas

El objetivo de esta sección es detallar la estimación del tiempo de dedicación a cada tarea del proyecto, la cual queda reflejada al final de este apartado en un diagrama de Gantt.

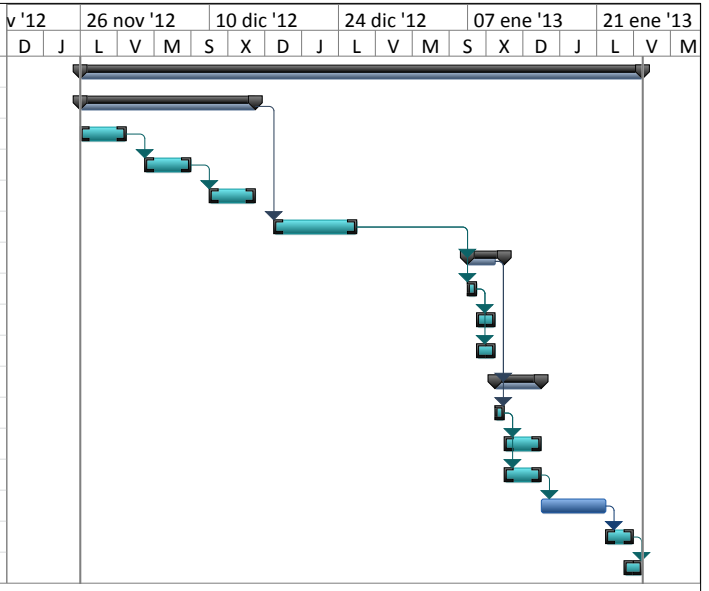
Como se adelantó en la sección anterior, el proyecto consta de tres partes independientes: el módulo de dibujo, el módulo de visión artificial y el módulo del algoritmo del juego. Cada uno de estos módulos equivale a una iteración del modelo en espiral y será totalmente independiente al resto.

El proyecto comenzó el 26 de noviembre de 2012 y terminó el 19 de junio de 2013, haciendo un total de seis meses de trabajo.

Para planificar el proyecto sólo se ha tenido en cuenta los días laborables ya que, al depender del robot que se encuentra en la universidad, no era posible avanzar en fin de semana, puentes o vacaciones. El desarrollo del módulo abarca desde el 26 de noviembre hasta el 25 de enero. Después se desarrolló el de visión, del 28 de enero hasta el 19 de marzo. Posteriormente, el módulo relacionado con el algoritmo de búsqueda, del 20 de marzo al 29 de abril. Finalmente, las tres semanas siguientes se dedicaron a la integración de todos los componentes, se grabaron las demos del juego y se realizaron las pruebas de integración y de sistema. De forma paralela se comenzó a documentar el proyecto. La semana del 3 de junio se aprovechó para realizar la evaluación del sistema con personas ajenas al proyecto.

El diagrama de Gantt que se muestra a continuación contiene todas las tareas llevadas a cabo durante el proyecto. Por comodidad y para llegar a un mayor nivel de detalle, el diagrama ha fragmentado en tres partes, representando cada parte cada uno de los tres módulos del proyecto (figuras 5.1, 5.2 y 5.3). Al final de esas tres figuras, se muestra el diagrama de Gantt general del proyecto (figura 5.4).

| Id | Nombre de tarea | Duración | Comienzo | Fin | Predecesoras |
|---|---|---|---|---|---|
| 1 | **MÓDULO DE DIBUJO** | **45 días** | **lun 26/11/12** | **vie 25/01/13** | |
| 2 | **Análisis de las alternativas** | **15 días** | **lun 26/11/12** | **vie 14/12/12** | |
| 3 | Kinect | 5 días | lun 26/11/12 | vie 30/11/12 | |
| 4 | Android | 5 días | lun 03/12/12 | vie 07/12/12 | 3 |
| 5 | Choregraphe | 5 días | lun 10/12/12 | vie 14/12/12 | 4 |
| 6 | **Estudio de símbolos a dibujar** | **7 días** | **lun 17/12/12** | **mar 25/12/12** | **2** |
| 7 | **Dibujo del símbolo X** | **4 días** | **lun 07/01/13** | **jue 10/01/13** | **6** |
| 8 | En una casilla | 1 día | lun 07/01/13 | lun 07/01/13 | 6 |
| 9 | En todas las casillas | 2 días | mar 08/01/13 | mié 09/01/13 | 8 |
| 10 | Acciones adicionales | 2 días | mar 08/01/13 | mié 09/01/13 | 8 |
| 11 | **Dibujo del símbolo cuadrado** | **3 días** | **jue 10/01/13** | **lun 14/01/13** | **7** |
| 12 | En una casilla | 1 día | jue 10/01/13 | jue 10/01/13 | 7 |
| 13 | En todas las casillas | 2 días | vie 11/01/13 | lun 14/01/13 | 12 |
| 14 | Acciones adicionales | 2 días | vie 11/01/13 | lun 14/01/13 | 12 |
| 15 | **Refinamiento del dibujo** | **5 días** | **mar 15/01/13** | **lun 21/01/13** | **14** |
| 16 | **Codificación en Python** | **3 días** | **mar 22/01/13** | **jue 24/01/13** | **15** |
| 17 | **Pruebas en el robot** | **2 días** | **jue 24/01/13** | **vie 25/01/13** | **16** |



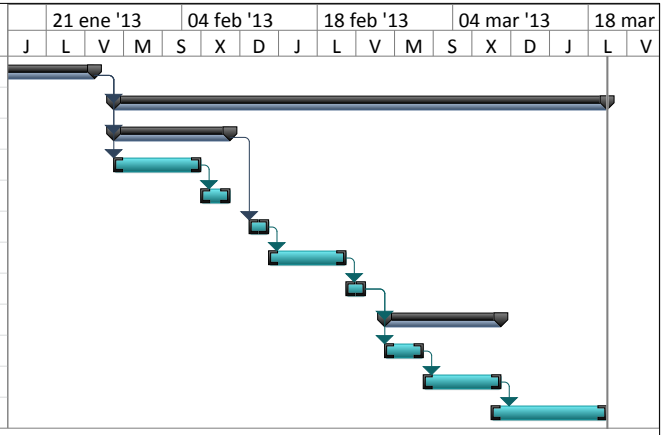

Proyecto: Módulo Dibujo
Fecha: dom 02/06/13

| | | | | | |
|---|---|---|---|---|---|
| Tarea | | Hito externo | | Informe de resumen manual | |
| División | | Tarea inactiva | | Resumen manual | |
| Hito | | Hito inactivo | | Sólo el comienzo | |
| Resumen | | Resumen inactivo | | Sólo fin | |
| Resumen del proyecto | | Tarea manual | | Fecha límite | |
| Tareas externas | | Sólo duración | | Progreso | |

Página 1

FIGURA 5.1: Tareas del Módulo de Dibujo

| Id | Nombre de tarea | Duración | Comienzo | Fin | Predecesoras |
|---|---|---|---|---|---|
| 1 | **MÓDULO DE DIBUJO** | **45 días** | **lun 26/11/12** | **vie 25/01/13** | |
| 18 | **MÓDULO DE VISIÓN** | **37 días** | **lun 28/01/13** | **mar 19/03/13** | **1** |
| 19 | **Análisis de las alternativas** | **10 días** | **lun 28/01/13** | **vie 08/02/13** | **1** |
| 20 | OpenCV | 7 días | lun 28/01/13 | mar 05/02/13 | 1 |
| 21 | Choregraphe | 3 días | mié 06/02/13 | vie 08/02/13 | 20 |
| 22 | **Obtención de imágenes con NAO** | **2 días** | **lun 11/02/13** | **mar 12/02/13** | **19** |
| 23 | **Obtención de histogramas** | **6 días** | **mié 13/02/13** | **mié 20/02/13** | **22** |
| 24 | **Comparación de histogramas** | **2 días** | **jue 21/02/13** | **vie 22/02/13** | **23** |
| 25 | **Integración de histogramas con NAO** | **10 días** | **lun 25/02/13** | **vie 08/03/13** | **24** |
| 26 | Estudio de alternativas | 4 días | lun 25/02/13 | jue 28/02/13 | 24 |
| 27 | Codificación de la solución | 6 días | vie 01/03/13 | vie 08/03/13 | 26 |
| 28 | **Integración de VISIÓN con DIBUJO** | **8 días** | **vie 08/03/13** | **mar 19/03/13** | **27** |



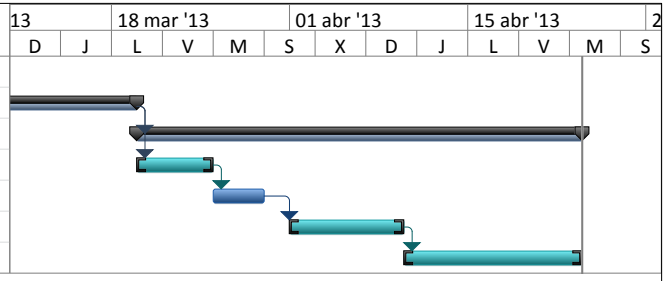

Proyecto: Módulo Visión
Fecha: lun 03/06/13

| | | | | | |
|---|---|---|---|---|---|
| Tarea | | Hito externo | | Informe de resumen manual | |
| División | | Tarea inactiva | | Resumen manual | |
| Hito | | Hito inactivo | | Sólo el comienzo | |
| Resumen | | Resumen inactivo | | Sólo fin | |
| Resumen del proyecto | | Tarea manual | | Fecha límite | |
| Tareas externas | | Sólo duración | | Progreso | |

Página 1

FIGURA 5.2: Tareas del Módulo de Visión

| Id | Nombre de tarea | Duración | Comienzo | Fin | Predecesoras |
|---|---|---|---|---|---|
| 1 | **MÓDULO DE DIBUJO** | **45 días** | **lun 26/11/12** | **vie 25/01/13** | |
| 18 | **MÓDULO DE VISIÓN** | **37 días** | **lun 28/01/13** | **mar 19/03/13** | **1** |
| 29 | **MÓDULO DE ALGORITMO DEL JUEGO** | **25 días** | **mié 20/03/13** | **mar 23/04/13** | **18** |
| 30 | **Búsqueda de algoritmo minimax** | **4 días** | **mié 20/03/13** | **lun 25/03/13** | **18** |
| 31 | **Análisis de alternativas encontradas** | **4 días** | **mar 26/03/13** | **vie 29/03/13** | **30** |
| 32 | **Adaptación de librería AIMA** | **7 días** | **lun 01/04/13** | **mar 09/04/13** | **31** |
| 33 | **Integración con DIB. VIS.** | **10 días** | **mié 10/04/13** | **mar 23/04/13** | **32** |



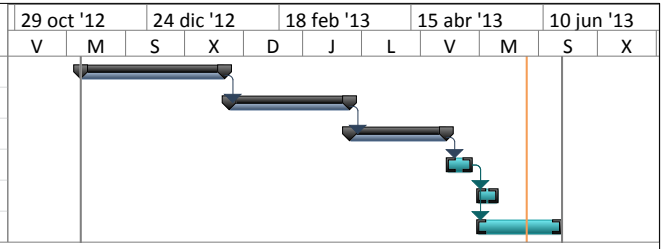

Proyecto: Módulo Algoritmo
Fecha: lun 03/06/13

| | | | | | |
|---|---|---|---|---|---|
| Tarea | | Hito externo | | Informe de resumen manual | |
| División | | Tarea inactiva | | Resumen manual | |
| Hito | | Hito inactivo | | Sólo el comienzo | |
| Resumen | | Resumen inactivo | | Sólo fin | |
| Resumen del proyecto | | Tarea manual | | Fecha límite | |
| Tareas externas | | Sólo duración | | Progreso | |

Página 1

FIGURA 5.3: Tareas del Módulo de Algoritmo del Juego

| Id | Nombre de tarea | Duración | Comienzo | Fin | Predecesoras |
|---|---|---|---|---|---|
| 1 | **MÓDULO DE DIBUJO** | **45 días** | **lun 26/11/12** | **vie 25/01/13** | |
| 18 | **MÓDULO DE VISIÓN** | **37 días** | **lun 28/01/13** | **mar 19/03/13** | **1** |
| 29 | **MÓDULO DE ALGORITMO DEL JUEGO** | **29 días** | **mié 20/03/13** | **lun 29/04/13** | **18** |
| 34 | **PRUEBAS** | 9 días | mar 30/04/13 | vie 10/05/13 | 29 |
| 35 | **EVALUACIÓN** | 7 días | lun 13/05/13 | mar 21/05/13 | 34 |
| 36 | **DOCUMENTACIÓN** | 26 días | lun 13/05/13 | lun 17/06/13 | 34 |




Proyecto: Diagrama General
Fecha: lun 03/06/13

| | | | | | |
|---|---|---|---|---|---|
| Tarea | Hito externo | Informe de resumen manual |
| División | Tarea inactiva | Resumen manual |
| Hito | Hito inactivo | Sólo el comienzo |
| Resumen | Resumen inactivo | Sólo fin |
| Resumen del proyecto | Tarea manual | Fecha límite |
| Tareas externas | Sólo duración | Progreso |

Página 1

FIGURA 5.4: Diagrama de Gantt general

## 5.2. Presupuesto

El presupuesto del proyecto está dividido en varios capítulos. En las siguientes secciones se calcula la parte del presupuesto relativa a los costes directos, los cuales incluyen el personal y el equipo hardware y software con el que se ha desarrollado el proyecto. Para los costes indirectos se ha fijado una valor del 20 % de los costes directos .

Para obtener el coste total de personal que ha supuesto el proyecto se ha fijado el coste/hora en función del puesto de trabajo. La tabla 5.1 recoge los costes relativos al personal.

| Categoría | Coste / hora | Horas | Coste (Euros) |
|---|---|---|---|
| Analista | 25 | 140 | 3.500 € |
| Desarrollador | 25 | 290 | 7.250 € |
| Jefe de proyecto | 35 | 70 | 2.450 € |
| Tester | 20 | 20 | 400 € |
| | | **TOTAL** | 13.600 € |

TABLA 5.1: Costes directos de personal

Respecto a los costes relacionados con el equipo utilizado, hay que calcular la amortización de cada uno de ellos. La fórmula que ha utilizado es la siguiente:

$$(A/B) * C$$

- A: número de meses que se ha utilizado el material en cuestión. En este caso 6 meses.
- B: número de meses de vida del material en cuestión.
- C: coste unitario del material en cuestión.

En la tabla 5.2 se reflejan todos estos costes donde la columna de coste imputable equivale al valor amortizado.

| Concepto | Coste Unitario (Euro) | Coste imputable (Euro) |
|---|---|---|
| Ordenador (4GB RAM 2,4 GHz) | 950€ | 95€ |
| Robot NAO H25 | 12.000€ | 1.200€ |
| Monitor 25” | 250€ | 25€ |
| | **TOTAL** | 1.320€ |

TABLA 5.2: Costes directos de equipo

Finalmente, se establece una tasa del 20 % sobre los costes directos que equivalen a los costes indirectos, los cuales son las facturas de la luz, el agua, el teléfono, el acceso a Internet etc.

La tabla 5.3 muestra el presupuesto total del proyecto, siendo éste de 17.904 €.

| Presupuesto Costes Totales | Presupuesto Costes Totales |
|---:|---:|
| Personal | 13.600€ |
| Equipo | 1.320€ |
| Costes Indirectos (20 %) | 2.984€ |
| **TOTAL** | 17.904€ |

TABLA 5.3: Presupuesto total del proyecto

# Capítulo 6

# Conclusiones y Líneas Futuras

En este capítulo se recogen las conclusiones del trabajo realizado, evaluando los resultados obtenidos y comparándolos con las metas inicialmente propuestas. En la sección 6.4 se habla de los resultados obtenidos en general del proyecto teniendo en cuenta las experiencias pasadas, los conocimientos adquiridos y la satisfabilidad del producto desarrollado. En la sección 6.1 se analizan los objetivos que se han cumplido y se justifican los que no ha sido posible cumplir o aquellos en los que se ha cambiado algo por otra cosa similar. En la sección 6.2 se resumen los problemas que se han ido encontrando a lo largo del trabajo y cómo se resolvieron. Por último, en la sección 6.3 se proponen mejoras aplicables al sistema desarrollado y nuevas líneas de investigación. Se toma como referencia las sugerencias recopiladas en el capítulo 4.

## 6.1. Conclusiones referentes a los objetivos

En la sección 1.2 se fijaron tres objetivos:

- El robot aprenderá a dibujar sobre un tablero.
  Este objetivo se ha cumplido gracias al software Choregraphe con el que se grabaron las instancias de entrenamiento que sirvieron para crear el comportamiento definitivo. Así se ha conseguido que NAO pueda dibujar el símbolo X en cualquier casilla. Como se ha probado en un entorno real, no hay duda de su funcionamiento y aunque el desarrollo de este tipo de comportamientos es costoso, se pueden diseñar nuevos para que NAO dibuje otros símbolos diferentes, pues el mecanismo es el mismo.

- El robot será capaz de identificar en qué situación del juego se encuentra y la casilla que ha escogido su rival.
  El uso de la librería OpenCV y las cámaras del robot fueron los responsables del desarrollo exitoso de este objetivo. Se descubrió el gran potencial de los histogramas ya que

permitieron crear un sistema de visión de baja dificultad y que obtenía una alta tasa de acierto.

- El robot será capaz de tomar la decisión sobre qué casilla pintar en cada turno.
  Este objetivo también se cumplió mediante la inclusión de un algoritmo minimax con una poda alfa-beta. Este algoritmo resultó ser imbatible durante la evaluación del sistema, pues ninguna persona fue capaz de ganar al robot. Aún así hay que destacar que el abanico de posibilidades en el juego del Tres en Raya es bastante reducido y esto facilita el análisis de las distintas decisiones que se quieran tomar.

El éxito de estos tres objetivos queda verificado además en el capítulo 4 tanto en las pruebas de sistema como en la evaluación realizada por personas ajenas al desarrollo del trabajo.

## 6.2. Problemas encontrados

A continuación se enumeran y explican los principales problemas que surgieron durante el desarrollo del sistema Tres en Raya.

- La traducción de movimientos del humano al robot para que éste consiguiera dibujar fue uno de los procesos más complicados y lentos del trabajo. A pesar de que la herramienta Choregraphe grababa cada uno de los movimientos, al reproducirlos en la mayoría de los casos se cometían errores de precisión, o bien porque no se apartaba el rotulador a tiempo o bien porque la línea salía curvada (por ejemplo en el caso de dibujar una X). Esto se resolvió dividiendo el dibujo del símbolo en partes atómicas, en concreto en dos diagonales independientes. De esta forma se reducía el tiempo de grabación y los movimientos podían corregirse mejor a posteriori.

- El segundo problema fue la curva de aprendizaje del lenguaje Python, pues aunque no es muy elevada y la sintaxis es sencilla, acostumbrarse a utilizar un lenguaje no tipado, que no exige la colocación de signos de puntuación al final de las líneas pero sí es vital la tabulación del código, hace un poco complicada la depuración de errores.

- El último problema al que aún no se le ha podido dar solución fue el error que comete el robot al desplazarse hacia la izquierda y la derecha para pintar en diferentes casillas. El robot curva levemente su trayectoria a pesar de que no se le ha indicado que lo haga y esto complica que se coloque en la posición correcta de forma autónoma.

## 6.3. Líneas Futuras

En la realización de este trabajo quedan abiertas muchas líneas de investigación y utilización de otras técnicas que podrían mejorar notablemente el funcionamiento del sistema y optimizar la duración de cada partida.

El dibujo de símbolos se podría mejorar utilizando aprendizaje por refuerzo para que la persona que actúa como "maestro" pueda, además de enseñar al robot, evaluar la calidad del dibujo de forma que lo haga cada vez mejor.

Por otro lado, respecto a la visión, se podría realizar un estudio de los valores obtenidos en la similitud de los histogramas para que según se obtengan los pares de imágenes a comprobar se generase la similitud y se pudiera detectar en ese mismo momento si esa casilla ha variado o no, pues los valores altos indican esto último. Así no haría falta analizar el tablero completo a no ser que la casilla que variase fuese la última (peor caso). De todas formas para que esto diese resultado se debería estudiar también la calidad de imágenes capturadas por el robot cuando se convierten a escala de grises y la precisión de éste al capturarlas en diferentes turnos, pues un leve movimiento provoca, en algunos casos, que el robot se equivoque. Otra idea relacionada con el reconocimiento de la casilla sería realizar la confirmación de que la casilla propuesta por el robot es correcta mediante reconocimiento de voz, evitando así utilizar la terminal de Linux y haciendo el sistema Tres en Raya más interactivo.

Otra vía alternativa al algoritmo de búsqueda sería aplicar un planificador como PELEA [28] que le fuera indicando en cada turno al robot el siguiente movimiento a partir de la información contenida en el tablero. Adicionalmente, también podría indicarle los desplazamientos y las órdenes como pintar símbolo y escanear tablero. En caso de sufrir alguna caída sería capaz de conseguir que el robot se recuperase y continuase jugando ayudado por la replanificación.

Relacionado con los desplazamientos del robot, el siguiente paso a realizar sería implementar un sistema que le permitiera al robot colocarse en la posición correcta frente a cada casilla para poder pintar sin necesidad de ser asistido por un humano. Esto se podría realizar utilizando visión artificial y algún símbolo similar a la NAOMark [1] con el que pudiera calcular la distancia a la que está colocado del tablero y si está situado de forma paralela a éste.

Finalmente, para mejorar el comportamiento social del NAO se podrían incluir ciertas frases con componente humorístico que se reprodujeran en función de la situación en la que se encuentre el juego. Así el comportamiento de NAO se asemejaría más al de un humano. Este propuesta se ha obtenido de las sugerencias de la evaluación del trabajo.

[1] http://www.youtube.com/watch?v=c3LrNcDuydM

## 6.4. Conclusiones Personales

Con la realización de este proyecto me he adentrado en el mundo de la robótica, que cada vez tiene más adeptos y más líneas de investigación. En concreto, el trabajo se sitúa entre el aprendizaje, la visión artificial y la algoritmia.

Como la robótica era un tema que no se había profundizado en las asignaturas del Grado en Ingeniería Informática, considero que he aprendido muchísimo, desde las nociones básicas como son el funcionamiento de sensores y actuadores hasta arquitecturas mucho más complicadas, como la del robot NAO.

Por otro lado, he aprendido a desarrollar programas en lenguaje Python. Ha sido la primera toma de contacto y reconozco que es un lenguaje potente y fácil de utilizar. Además cada vez más personas eligen este lenguaje para desarrollar sus aplicaciones y esto añade mucho más valor al conocimiento adquirido (figura 6.1).

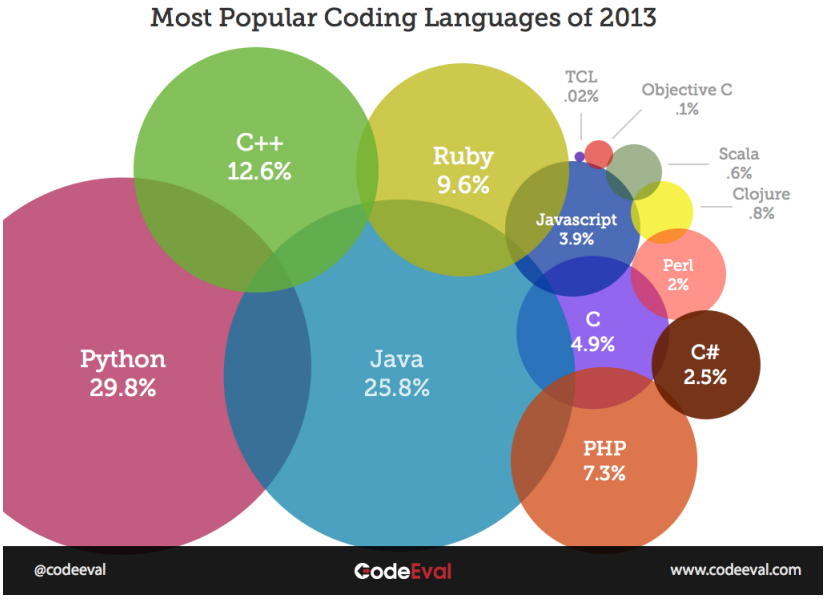



FIGURA 6.1: Lenguajes de programación más utilizados en 2013. (Fuente)

Respecto a trabajar con el robot NAO, el conocimiento adquirido tiene mucho valor ya que es uno de los robots más vendidos a nivel mundial en los últimos años. Utilizar el software Choregraphe y NaoQi me han proporcionado grandes habilidades y también me han permitido conocer de primera mano lo que supone trabajar con un robot, sobre todo a lidiar con los riesgos: seguridad, imprevistos, inestabilidad etc.

En último lugar, me gustaría destacar que la elaboración de esta memoria se ha hecho íntegramente en LaTeX. Aunque ya lo había utilizado el año anterior en pequeños trabajos, el desarrollar esta memoria, especialmente larga, utilizando esta tecnología me ha supuesto conocer un gran

número de paquetes nuevos y me ha permitido mejorar la calidad de la documentación, como por ejemplo, incluyendo imágenes vectorizadas.

# Capítulo 7

# Developing a Tic Tac Toe playing system for robot NAO H25

## 7.1. Introduction

We live in a society where technology moves on faster than ever and new devices such as smartphones, computers, game consoles, 3D TVs etc. are launched frequently. As a result, innovation has been transformed into an essential target for companies in order to keep gaining large amounts of profits.

We expect our life to be more comfortable and enjoyable each day and this turns into reality with the evolution of technology. We are used to live surrounded by electronic devices and constantly surfing on the Internet. Thus, it is common to have in home an intelligent washing machine, a cleaning or cooking robot, etc. or receiving on our virtual mail box personalized recommentations of products. Not only are we surrounded by these kind of intelligent devices in home but they are also taking place in almost everywhere: hospitals, shopping centers, airports, etc.

The majority of these devices, objects or web servicies mentioned before, have a huge backend built with different artificial intelligent tecniques. It is the only way known to modify in real-time the behavior of a program. For example, it helps to avoid that the cleaning robot hits a wall itself, or that a virtual agent can interact with a human by using its voice or by writing text messages.

Unfortunately, in robotics, the cost and complexity of the development of a robot wich includes all these features is very high. Hardware evolves faster than software, so that we can easily see hardy but barely intelligent robots. Tipically, robots are designed to solve a finite number of tasks in an autonomous way or collectively with other robots. Something as easy as standing up and

sitting down without any help can be a huge obstacle for a biped robot. As a consequence, these limitations affect directly on the way of programming a complex behavior. Research lines in robotics are focused mainly on decreasing the technological gap in order to get better intelligent or human-like behaviours, but each particular case has to be analized individually. Nowadays, the closest human-like robots are social robots because of their comunicative skills with the environment.

This idea of decreasing the technological gap between hardware and software is studied in this work to come up with a solution for a specific task. A robot may not have been built for doing some tasks, but that does not mean that it can not do them. If you provide robots with the techniques as the ones that AI offers, the robot could be able to learn. Even if its actions are not as perfect as the actions of another robot specialized in the field, your robot will be more versatile. This aspect is important when working with social robots, as it is very interesting for robots to know how to solve a multitude of tasks and situations in order to manifest a closer relationship with humans.

## 7.2. Main Objectives

This work includes the development of a system designed for the humanoid robot NAO H25. The purpose of this system is to make the robot play, autonomously, Tic Tac Toe against a human player by drawing both on a board.

First, the robot will learn to draw two different symbols on a board. Then, it will be able to identify through vision what the game situation is and which cell has chosen its rival. Finally, the robot will be able to take the decision about what cell to play at each turn.

Note that this robot is not specialized in precision tasks such as drawing, so this increases the difficulty of the development of the system.

## 7.3. Architecture

In this section, the architecture of the system is explained. It has been divided in three different modules: Drawing, Computer Vision and Algorithm. Each one of them has several functionalities which are essential for the correct performance of the system.

Figure 7.1 shows how modules are connected in the global architecture. The output of a module is the input of another one. The diagram has been designed from the point of view of the robot, as the human player only has one task: choosing the next move.

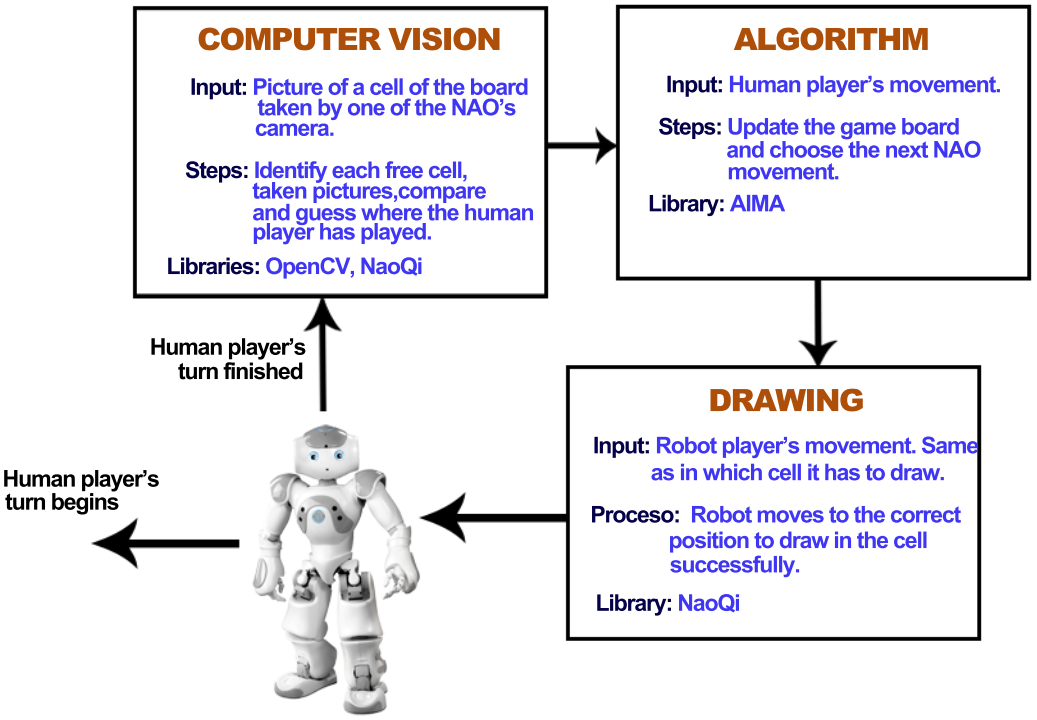



FIGURA 7.1: Global architecture of the Tic Tac Toe playing system.

Starting from the end of the human player turn, the robot will analize each cell of the board by taking pictures with its two cameras. When the robot has finished, it will compare the pictures with the ones taken in the previous turn. To do that, the system extracts the histogram of each couple of pictures and applies to them a similarity test by using the Chi-Square distance. The highest value of all the pairs of images will be the cell the human player has chosen. Once the system knows that information, the Minimax algorithm helped by the Alpha-Beta prunning explores the available movements on the board and then, it will tell the robot the most beneficial option for him to win the game. Finally, the robot will draw, thanks to the drawing actions of the Drawing module, in the cell that matches with the movement.

For better understanding how the system works, figure 7.2 presents a flow diagram of the global architecture.

The game starts by picking up randomly the first player to draw on the board. Then, either if the first player is the robot or the human, the system will check if there is a winning move. If that is true, the game will stop and the robot will announce the winner.

The robot turn is not as easy to play as the human's because of the number of tasks that the robot should perfom. That is why figure 7.3 shows a flow diagram dedicated exclusively to the development of the robot turn.

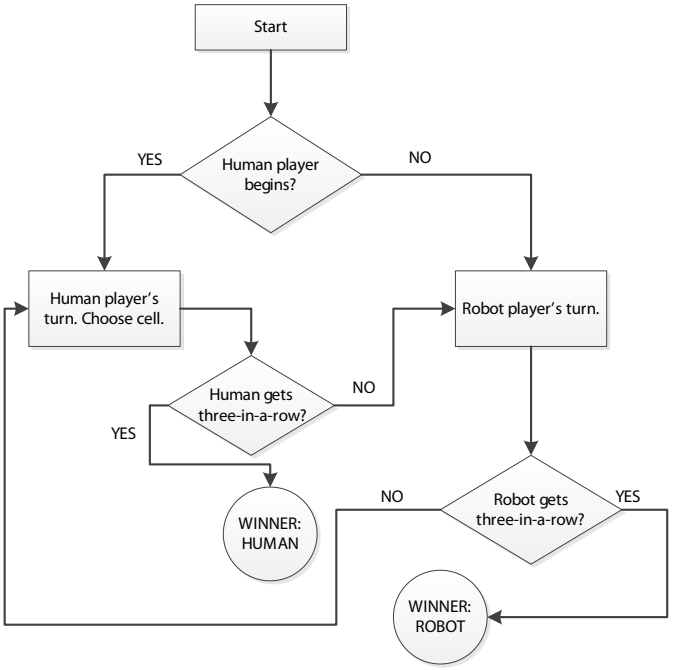



FIGURA 7.2: Flow diagram of the Tic-Tac-Toe playing system.

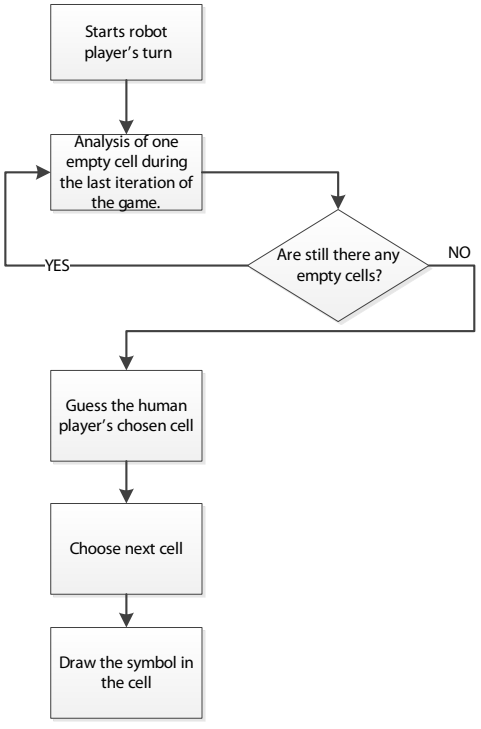



FIGURA 7.3: Flow diagram of the robot's turn.

## 7.4. Drawing module

This module has all the actions that allow NAO to draw a symbol in any cell of the board. These actions are divided in Poses, Movements and Draw of Symbols.

On the Poses group, we have used the following actions: *Stand-Up*, *Sit-Down*, *Pose-Init* and *Fall-Down Recovery*. They all have in common that they were included on the robot before this system was developed. Thank to *Stand-Up* and *Sit-Down* the robot can get up and down on the floor. The *Pose-Init* action prepares the robot in the correct position (figure 3.15) to start drawing or moving without taking any risk. Finally, the *Fall-Down Recovery* action, as the name says, helps NAO to recover from an unexpected fall.

Then, the Movements module is formed by all the actions related to changing the current position of the robot. Move to the right, left, straight or back are four actions included in the module followed by getting up and down. Depending on the cell where the robot has to move, it will use a bigger o smaller set of these actions. After the robot drew, it must come back to the initial position, so that it will have to execute the actions backwards.

The last group of actions of the Drawing Module includes everything related with drawing: placing the marker, draw a symbol and put down the marker are the main actions. By using placing the marker, the robot will put the marker in the correct position to start drawing. Putting down the marker it is the opposite and it will be used after drawing. Finally, drawing a symbol will let the robot draw an X or a square on a cell of the board.

About the initial position, it was decided to be the position in which NAO could draw a symbol in cell number three. That is because the robot is placed in the middle of the board so that it is the best position to do later the scan of the board. We have established some references for setting the robot in this position: NAO should stay at 8 cm of the wall. Also its feet should be in the DIN-A4 paper so that they matched its footprints (figure 3.25). The last reference is about what NAO can see in that position; the cell number five with the camera of this head and the cell number eight with the camera of his mouth (figure 3.24).

## 7.5. Computer Vision Module

The aim of the Computer Vision Module is to identify which cell the human player has chosen in his/her turn. It is divided in two parts: taking pictures and analyzing pictures taken.

The robot will capture an image of the cell only if it was empty during the last robot turn. This process is repeated for each cell of the board. To capture an image, the robot has to take first a set of actions that include move its head (up, down, left or right), get down, get up and access to its two cameras. Depending on the cell the robot wants to take the image from, it will use a bigger o smaller set of these actions.

Once all the pictures have been taken, the system generates each histogram by calling the fuction histogram from the OpenCV library. Then, the function *imread*[1] will convert the image into a Grayscale one.

The next step is to use the fuction *calcHist* [2] which returns the histogram of the image in a list of values. Each one of them equals the quantity of pixels of the image that have the same luminosity. Finally, each couple of images is compared with the function *compareHist* [3], also from OpenCV library. The similarity value is calculated by using the Chi-square distance, in which the higher the value, the lower the similarity of the histograms, so that the cell that the human player has chosen will be the highest value of the list of similarity values. The Chi-Square distance has been calculated with this formula:

$$d(H_1, H_2) = \sum_{I=0}^{256} \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

- $H_n$: n histogram
- $H_n(I)$: Value of the I bar of the n histogram

## 7.6. Algorithm Module

The aim of this module is to figure out which is the next best move for NAO taking into account the situation of the game. We have used the Minimax algorithm with Alpha-Beta prunning included in the AIMA-python library [26].

This module receives as an input the move chosen by the human player so the first thing to do is updating the virtual board. To do that, as the names of the cells are different in the real board and the virtual one, the input has to be transformed. Table 7.1 shows the equivalences between boards.

[1] http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html#imread

[2] http://docs.opencv.org/modules/imgproc/doc/histograms.html#calchist

[3] http://docs.opencv.org/modules/imgproc/doc/histograms.html#comparehist

| Real board | Virtual board |
|---|---|
| 1 | (1, 1) |
| 2 | (1, 2) |
| 3 | (1, 3) |
| 4 | (2, 1) |
| 5 | (2, 2) |
| 6 | (2, 3) |
| 7 | (3, 1) |
| 8 | (3, 2) |
| 9 | (3, 3) |

TABLA 7.1: Equivalence between boards.

Now the board can be updated with the function *result()* from the AIMA library. We have also used the display() function to show the virtual board on the terminal. The next example shows what can be seen in the terminal when the input has been received and converted.

```
--------------------
   It's number...6
--------------------
. . .
. . X
. . .
```

The next step involves calling the algorithm Minimax (sectión 2.4) and the Alpha-Beta prunning (sectión 2.5). The system calls the *alphabeta_search()* to expand the nodes of the tree generated by the Minimax algorithm. The root node is the current situation of the board and each expanded node means that a new move has been made on the board. When the board is full or the maximum depth has been reached, leaf nodes will be analyzed and given a value depending on the situation of the board in that node:

- -1 if human player wins
- 0 if it is a draw
- 1 if robot wins

These values will go backwards through the branches of the tree until one of them reaches the root node. The returned values depend on the kind of node of their parent. A MAX node will return the maximum value of its children. A MIN node will return the minimum value. Once the root node knows the successor with the maximum value, it will return that move to the robot,

which will be the one it has to draw on. The last step is to update the virtual value before letting the robot draw on the real board.

When the algorithm finishes the search for the next movement, the system will print on the terminal the next fragment of text. In the example, NAO draws the symbol on the cell number three. Before the robot starts to draw, it will say to the human player that its turn begins. Once it has finished drawing, it will say "Human player it's your turn!". This can be done by using the function *talk()*.

```
--------------------
NAO it's your turn!
--------------------
(1, 3)
. . O
. . X
. . .
```

## 7.7. Related Work

In this section we describe three works which also have developed a Tic Tac Toe playing system for the robot NAO. In all of them the robot plays against a human player.

The first work has been developed by Franck Calzada. The system is developed on Python and the author has used Choregraphe to create new behaviours and OpenCV library to provide a real-time computer vision module to the system. The board has been placed over a table.

NAO can detect when there is a board placed in front of him to play and when the board is full and it can't play anymore. If the robot can not see the board well, it will complain by saying out loud to the human to change the position of the board. The robot also talks when its turn begins or ends, when it needs a marker to draw and when human turn begins or ends.

On the other hand, NAO analyzes the board in order to choose the next cell and win the game. The symbol that NAO draws in each cell is a dash. The next video shows how this system works in a real environment.

http://www.youtube.com/watch?v=p1ITwOEZAdA

The second work it is a bit different of the previous one because the board is placed in a wall. The robot uses computer vision in order to identify the situation of the game and choose the next movement. In this case, instead of drawing a little dash, NAO takes a magnetic piece and place it on the chosen cell. To pick up the pieces, the robot asks for help and opens its hand; immediately, a human will provide the magnetic piece to the robot. The next video shows how this system works in a real environment.

http://www.youtube.com/watch?v=WLemr3WspXE

The last work presents a similar environment to the previous one because the board is placed in a wall and both players play with pieces instead of drawing them. There is a notably difference; the robot NAO is not autonomous. It only says where it wants to put the piece to a human and he will do all the work for the robot. Despite this, the system has, as the others, computer vision and a decision-making module. The next video shows how this system works in a real environment.

http://www.youtube.com/watch?v=DLgqzmEdw_I

## 7.8. Conclusions and Future Work

### 7.8.1. Conclusions

Three goals were established at the start of this work:

- The robot will learn to draw on a board.
  This goal was reached thanks to the software Choregraphe. Instances of training were recorded on it in order to create the new behaviours. As a consequence, NAO can draw X symbol in any cell of the board. This system has been tested in a real environment. Despite of the fact that the development of this kind of behaviours is slow and difficult, using the same tools can be designed any other behaviours so that NAO will be able to draw other symbols on the board.

- The robot will be able to identify what the game situation is and the cell that has chosen his rival.
  OpenCV and NAO's cameras were responsible for the successful development of this goal. By using the potential of histograms an easy and really effective computer vision system was created. In addition, the system has a high rate of success in finding which cell the human player chose.

- The robot will be able to take the decision about in which cell to play at each turn.
  We reached this goal by using the Minimax algorithm with Alpha-Beta prunning. This algorithm proved to be unbeatable in the system evaluation, as no human was able to win the robot. Fact partially happened because of the reduced range of possibilities in Tic Tac Toe, which makes the analysis of decisions made easier.

The success of these three goals was verified in chapter 4 both in system tests as in the evaluation made by people outside the development of this work.

### 7.8.2. Future Work

There are many research lines that still open and many other techniques which could improve the performance of the system.

NAO's drawing skill could be improved by using reinforcement learning so that the person who plays the teacher role could reward the quality of the drawing. Using this information, NAO will draw better each time.

On the other hand, a study on the histograms similarity could be done, in order to detect when a high value obtained is enough to stop scanning the board. That would notably reduce the duration of the game unless the chosen cell is the number nine (worst case). Another option is to study the quality of the pictures taken by the robot when they are converted to a Grayscale. Related to the hit rate, a way to improve it is to analyze the little movements that can be seen in the couple of pictures and detect the ones that create a failure. Another alternative is to use voice recognition so the human player can confirm the robot if it has hit or failed the cell instead of using the keyboard. That would make the system more interactive.

Related to the robot movements, the next step would be to develop a system which allows the robot to place itself in the correct position in front of each cell autonomously. This can be don by combining computer vision with some marks, such as the NAOMark [4], so that the robot can calculate the distance between the board and itself.

Finally, to improve the social behaviour of NAO, isome funny sentences could be included, so that the robot could use them depending on the situation of the game. By doing that, NAO could look more like a human. This last proporsal has been taken from the suggestions of the evaluation of the work.

[4] http://www.youtube.com/watch?v=c3LrNcDuydM

# Bibliografía


[1] J. Ruiz del Solar and R. Salazar. *Introducción a la robótica*. Introducción a la robótica. URL http://robotica.li2.uchile.cl/EL63G/.

[2] INTEFP. Sistemas de control de lazo abierto y cerrado. URL http://recursostic.educacion.es/secundaria/edad/4esotecnologia/quincena11/4quincena11_contenidos_2a.htm.

[3] Frida Expósito. Tipos de robots, 2010. URL http://informaticafrida.blogspot.com.es/2009/03/tipos\discretionary{-}{}{}de\discretionary{-}{}{}robots.html.

[4] Moisés Martínez. Cómo dominar a un robot. URL http://prezi.com/pj1hovnzwj0o/como-dominar-un-robot/.

[5] Oscar Padilla Montiel. *Teleoperación*. Manipulador teleoperado inalámbricamente. 2008. URL http://catarina.udlap.mx/u_dl_a/tales/documentos/lmt/padilla_m_o/capitulo2.pdf.

[6] Tecnun. ¿qué es la teleoperación? URL http://www.tecnun.es/asignaturas/control1/proyectos/teleop2D/teleoperacion.htm.

[7] Rafael Rivas Estrada. Diseño software de una arquitectura de control de robots autónomos inteligentes. aplicación a un robot social, 2010. URL http://e-archivo.uc3m.es/bitstream/10016/9472/1/tesisRafaelRivasEstrada-Julio2010.pdf.

[8] Antoni Escrig Vidal. Control de movimiento de robots en presencia de incertidumbre: un enfoque adaptativo, 2006. URL http://coitt.es/res/revistas/Antena164_06A_Control_movimiento.pdf.

[9] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Dynamics and Control*. Second edition edition, 2004.

[10] Universidad de Málaga. Linealización del modelo dinámico, 2011. URL https://intranet.isa.uma.es/wiki/index.php?title=Linealizaci%C3%B3n_del_modelo_din%C3%A1mico.

[11] Epistemowikia. Lógica borrosa y robótica. URL http://campusvirtual.unex.es/cala/epistemowikia/index.php?title=L%C3%B3gica_Borrosa_y_Rob%C3%B3tica.

[12] Anis Fatmi, Amur Al Yahmadi, Lazhar Khriji, and Nouri Masmoudi. A fuzzy logic based navigation of a mobile robot. In *World Academy of Science, Engineering and Technology 22*, 2008. URL http://www.waset.org/journals/waset/v22/v22-183.pdf.

[13] Alessandro Saffiotti. Fuzzy logic in autonomous robot navigation: a case study. Technical report, 1997. URL http://fmi.uni-sofia.bg/courses/biomed/robots/ITsolutions/hfc98.pdf.

[14] E. Aguirre, J. C. Gómez, and González. A. A multi-agent system based on fuzzy logic applied to robocup's environment. 8(Mathware and Soft Computing): 153–178, 2001. URL http://ic.ugr.es/Mathware/index.php/Mathware/article/viewFile/163/141.

[15] Ana Beatriz Solana Sánchez. Juegos con adversario: Algoritmo minimax. URL http://www.it.uc3m.es/jvillena/irc/practicas/estudios/minimax.pdf.

[16] Grupo de Planificación y Aprendizaje (PLG). Búsqueda. URL http://ocw.uc3m.es/ingenieria-informatica/inteligencia-artificial-2/material-de-clase-1/minimax.pdf.

[17] Santiago Alfaro Ballesteros. Sistema de teleoperación mediante una interfaz natural de usuario, 2012. URL http://e-archivo.uc3m.es/bitstream/10016/16682/1/PFC_Santiago_Alfaro_Ballesteros.pdf.

[18] Aldebaran Robotics. Aldebaran roboticsófficial website, . URL http://www.aldebaran-robotics.com/en/Pressroom/About/Aldebaran.html.

[19] Aldebaran Robotics. The nao league, . URL http://www.aldebaran-robotics.com/en/Solutions/For-Robocup/the-NAO-league.html.

[20] The Robocup Federation. Soccer standard platform league. URL http://www.robocup.org/robocup-soccer/standard-platform/.

[21] Aldebaran Robotics. Choregraphe, 2013. URL http://www.aldebaran-robotics.com/en/Discover-NAO/Software/choregraphe.html.

[22] Aldebaran Robotics. Naoqi, 2013. URL http://www.aldebaran-robotics.com/en/Discover-NAO/Key-Features/NAOqi.html.

[23] Willow Garage. Robot operating system. URL http://www.ros.org/wiki/.

[24] Itseez. Opencv, 2013. URL http://opencv.org/.

[25] Stuart Russell and Peter Norvig. Artificial intelligence: A modern approach website, 2012. URL http://aima.cs.berkeley.edu/index.html.

[26] Peter Norvig. Aima-python, 2011. URL https://code.google.com/p/aima-python/.

[27] Barry W. Boehm. A spiral model of software development and enhancement. 1988. URL http://weblog.erenkrantz.com/~jerenk/phase-ii/Boe88.pdf.

[28] Cesar Guzman and Vidal Alcazar. Pelea: a domain-independent architecture for planning, execution and learning. In *Proceedings of ICAPS'12 Scheduling and Planning Applications woRKshop (SPARK)*, pages 38–45. AAAI Press, 2012. URL http://pst.istc.cnr.it/PlanSIG10/proceedings/paper%208.pdf.

# Apéndice A

# Manual de instalación

En este apéndice se describen los pasos para instalar cada una de las dependencias del Módulo Tres en Raya para un sistema operativo Ubuntu 11.04.

## A.1. Python

Si se ha instalado el sistema operativo Ubuntu 11.04 no habrá ningún problema con la versión de Python que se debe utilizar, pero por si surgiera alguna duda, se puede introducir este comando en una terminal:

```
python --version
```

En la pantalla aparecerá `Python 2.7.1+`. En caso de no disponer de Python 2.7 o 2.6 habrá que instalarlo en el sistema. Una versión superior no es válida por incompatibilidad con el software del robot NAO. El comando que permite instalar Python es el siguiente:

```
sudo apt-get install python2.6
```

## A.2. OpenCV

Para instalar OpenCV en un sistema operativo Ubuntu primero hay que descargarse una versión superior a la 2.4 desde el siguiente enlace [1].

Después hay que descomprimir el archivo descargado y comprobar si se encuentran instaladas en el sistema todas las dependencias. Para ello, en una terminal se escribe el siguiente comando:

[1] http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/

```
sudo apt-get install build-essential libgtk2.0-dev libjpeg-dev libtiff4-dev
libjasper-dev libopenexr-dev cmake python-dev python-numpy python-tk
libtbb-dev libeigen2-dev yasm libfaac-dev libopencore-amrnb-dev
libopencore-amrwb-dev libtheora-dev libvorbis-dev libxvidcore-dev libx264-dev
libqt4-dev libqt4-opengl-dev sphinx-common texlive-latex-extra libv4l-dev
libdc1394-22-dev libavcodec-dev libavformat-dev libswscale-dev
```

El siguiente paso es dirigirse a la ruta donde se ha descomprimido el archivo que contiene OpenCV y ejecutar los siguientes comandos para compilar la librería:

```
mkdir build
cd build
cmake -D WITH_TBB=ON -D BUILD_NEW_PYTHON_SUPPORT=ON -D WITH_V4L=ON -D
INSTALL_C_EXAMPLES=ON -D INSTALL_PYTHON_EXAMPLES=ON -D BUILD_EXAMPLES=ON
-D WITH_QT=ON -D WITH_OPENGL=ON ..
```

Por último, sólo queda ejecutar los comandos que instalarán la librería en el sistema:

```
make
sudo make install
```

## A.3. AIMA-Python

De esta librería tan sólo es necesario descargarse la parte dedicada a juegos, que se encuentra en *games.py*. Adicionalmente, como *games.py* llama a funciones de *utils.py*, también se necesitará descargar ese fichero. Los dos ficheros se encuentran en el siguiente enlace[2]. La librería AIMA no necesita instalación, para utilizarla tan sólo hay que escribir en el código Python *import nombre-del-fichero*.

## A.4. Choregraphe

El simulador Choregraphe puede obtenerse en el siguiente enlace para probarlo durante 30 días. Después se deberá introducir la licencia. Para instalarlo, es suficiente con descargar el archivo, descomprimirlo en el directorio donde se quiera utilizar y después ejecutarlo con el siguiente comando:

```
./choregraphe
```

[2] http://aima-python.googlecode.com/svn/trunk/

## A.5. NaoQi

Para instalar NaoQi una vez descargado, sólo hay que descomprimirlo en el directorio donde se quiera utilizar y después ejecutarlo con:

```
./choregraphe
```

Si se obtiene algún error relacionado con la variable de entorno PYTHONPATH a la hora de ejecutarlo, habrá que modificar la ruta asociada a la misma por la de Python que se encuentra dentro de la carpeta NaoQi.

```
$ export PYTHONPATH=${PYTHONPATH}:/path/to/python-sdk
```

# Apéndice B

# Manual de usuario

Una vez descargadas e instaladas todas las dependencias, en una carpeta incluiremos los ficheros *config.py* y *tres_en_raya.py*. Dentro de *config.py* hay que comprobar si la IP que aparece se corresponde con la que posee el robot, si esto no es así habrá que modificar la siguiente línea con la IP nueva

```
IP = ''192.168.1.100'' # set your Ip adress here
```

El puerto no hay que modificarlo; por omisión, siempre será el 9559.

Para ejecutar el módulo Tres en Raya, una vez que el robot está encendido y conectado a la red, tan sólo hay que escribir en una terminal lo siguiente:

```
python tres_en_raya.py
```

Lo primero que se observa por pantalla es que se ha establecido la conexión entre el PC y el NaoQi del robot. Equivale a las siguientes líneas:

```
-------------------------------------------
Loading proxy
-------------------------------------------
[INFO ] Starting ALNetwork
[INFO ] NAOqi is listening on 127.0.0.1:54012
-------------------------------------------
```

Después dará comienzo el juego en cuanto el robot NAO diga: *"Let's start playing"*. El primer paso siempre lo da NAO, realizando el primer escaneo del tablero y advirtiendo a su contrincante mediante la frase *"Scanning the board, please be patient"*. Seguidamente, el programa echará a

suertes quien de los dos jugadores empieza a jugar primero. El resultado lo comunicará NAO mediante una de las siguientes frases *'Human player you go first'* o *"I go first"*.

Cuando comience el turno del jugador humano, NAO se lo comunicará mediante la frase *"Human player it's your turn"*. También aparecerá por la terminal la misma frase:

```
-------------------------------------------
Human Player it's your turn!
-------------------------------------------
```

Entonces dicho jugador deberá pintar con el rotulador el símbolo escogido en la casilla deseada. Se recomienda que el símbolo se rellene para facilitar el reconocimiento por visión de NAO. Cuando se termine de pintar el símbolo y el jugador humano esté listo para pasar el turno al robot, deberá pulsar uno de sus *bumpers* (figura B.1). De esta forma, se le indica al robot que puede continuar con el juego.

FIGURA B.1: Situación de los bumpers de NAO

Cuando comience el turno del robot, lo primero que hará, siempre que antes haya jugado el humano, es escanear el tablero para analizar la situación en la que se encuentra el juego y averiguar cuál ha sido el último movimiento de su contrincante. Una vez analice todas las casillas y averigüe cuál ha sido el último movimiento, el programa lo mostrará en la terminal y será como el ejemplo siguiente.

```
------------------------------------------------------------------------
   Guessing cell...
------------------------------------------------------------------------
[(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)]
------------------------------------------------------------------------
   Results...
------------------------------------------------------------------------
{1: 10611.115446017697,
 2: 17587.099393067983,
 3: 3691.73786072385,
 4: 19949.4464614347,
 5: 90751.44339691453,
 6: 8326.877765692014,
 7: 24062.747603332464,
 8: 18899.78830623116,
 9: 39854.54058498882}
------------------------------------------------------------------------
   It's number... 5
------------------------------------------------------------------------
```

Después, el robot pedirá confirmación. A través de la terminal, el contrincante o cualquier otra persona de apoyo deberá contestar si o no en caso de que el robot haya acertado o no la última jugada del contrincante. Si el robot se ha equivocado deberá introducirse después por teclado el número de la casilla correcta. A continuación se muestran las preguntas y respuestas de ambos escenarios.

```
¿Es esta la casilla que has elegido? Contesta si/no  si
--
¿Es esta la casilla que has elegido? Contesta si/no  no
Introduce el valor numérico correcto de la casilla por favor  1
```

Una vez resuelto esto, el robot procederá a pintar en la casilla que desee y se volverá a repetir todo de nuevo hasta que uno de los dos jugadores consiga hacer Tres en Raya o bien queden en tablas. Cuando cualquiera de esos tres momentos se produzca en el juego, el robot será el encargado de comunicarlo mediante las frases:

- Si gana el humano
  *"Game Over. You win this time but it won't ever happen again"*.
- Si gana el robot
  *"Game Over. I am the winner. You are the loser"*.
- Si quedan en tablas
  *"Game Over. Draw. No one has won the game"*.

# Apéndice C

# Evolución del proceso de aprendizaje para dibujar símbolos

La técnica del aprendizaje por demostración es lenta y costosa, pues como ya se explicó en la sección 2.3.3, hay que recopilar numerosas instancias para poder generalizar al máximo el comportamiento que se quiere aprender.

En este apéndice se muestra el proceso que se siguió para conseguir que NAO dibujase símbolos en la pizarra (equis y cuadrado). La explicación irá acompañada de vídeos que reflejan perfectamente cómo fue mejorando la habilidad de dibujar de NAO.

En primer lugar se enseñó a NAO a situar el rotulador en una posición apta para dibujar. Como NAO sólo dispone de tres dedos no puede coger el rotulador como lo haría un humano así que se coloca perpendicular al brazo. En el primer vídeo se puede ver cómo se le enseña a NAO el movimiento que debe realizar y después la reproducción inmediata de éste. Tras grabar varios movimientos similares, se optimizó la velocidad y el leve balanceo del propio robot.

Situar rotulador

Después se comenzó a trabajar en la habilidad de dibujar. Repitiendo los mismos pasos, primero una persona le mostraba a NAO los movimientos que debía realizar y seguidamente NAO repetía el movimiento que había grabado. Como se podrá observar en los vídeos, al depender del rotulador y de la fuerza que NAO aplica sobre éste, era muy complicado obtener una instancia buena en los primeros experimentos. Por este motivo se grabaron numerosas instancias utilizando el propio pulso o una escuadra como herramienta auxiliar para trazar las diagonales. Así se consiguió que NAO dibujase equis y círculos, primero sobre un folio acolchado por detrás y después sobre la superficie del tablero.

Para optimizar el dibujo de los símbolos se probó a variar la velocidad en determinados tramos, a realizar el símbolo en dos partes, comenzar a dibujarlo desde otro lugar y pulir los errores al comenzar y terminar el símbolo (picos, líneas rectas etc).

NAO aprende a dibujar X

NAO dibuja símbolos en pizarra

NAO dibujar varias X

Cuando el desarrollo va avanzando y ya se dispone de tablero completo para jugar, la forma de dibujar un símbolo en la primera fila no es igual a tener que dibujarlo en la segunda o la tercera porque el robot debe agacharse. Por este motivo se volvieron a grabar instancias para estos dos casos y se repitió de nuevo todo el proceso. El resultado se puede ver en el siguiente vídeo.

NAO dibuja X en la tercera fila

Finalmente, se muestra en el último vídeo la habilidad de dibujar completamente pulida. NAO es capaz de dibujar una X manteniendo el trazado de las diagonales sin curvarse.

Versión final símbolo X